A comparison between ANNs and traditional stock market forecasting techniques

Alexandros Antoniou

Centre for Computational Finance and Economic Agents

CSEE

University of Essex

Submitted in partial satisfaction of the requirements for the
Degree of Master of Science
in
Computational Finance

Supervisor Dr Maria Kyropoulou
Second Supervisor Dr John O'Hara

September 2020

Acknowledgements

I would like to thank my supervisor, Dr Maria Kyropoulou, for helping me with this thesis. I would also like to thank my friends and family for keeping me sane during this lock-down and overall worldwide panic. It has not been a fun time.

Abstract

We employ LSTM models to fit and forecast daily closing prices for stocks and provide a comparison between the NN/LSTM method of modelling time-series and the more traditional method of using empirical observations on the structure of the data to construct and fit an ARIMA model. We collect daily stock data (open, high, low, close) from FTSE and NASDAQ companies over a range of 3 years and collect 1-day closing price forecasts from the two models for evaluation. Evaluation is performed on both models with their MSE and MAE values averaged and compared for a higher level overview of the two models. Results indicate that the LSTM model yields lower error values than the ARIMA model.

Keywords: Deep Learning, Artificial Neural Networks, Stock Market, Time Series prediction, Neural Networks

Table of Contents

1 Introduction

The ability to predict changes in stock prices is extremely important to the financial world as it influences trading strategies and reduces risks in the market. Forecasting has long been a problem for the business and technology communities and has seen few advances until quite recently, with the advent of neural networks and deep learning.

Before artificial neural networks, the finance world used other methods to model the time series that arose from the continuous updating of stock prices. Models like the autoregressive integrated moving average model (ARIMA) and the generalised autoregressive conditional heteroscedasticity model (GARCH) have become key econometric methods for forecasting time series and are still widely used in finance.

The focus of this project is to provide a comparison between the traditional methods for time series forecasting mainly the ARIMA model, and simple implementations of artificial neural networks, in the context of financial time series prediction.

Forecasting stock prices with moving averages belongs to the technical analysis category of financial analysis. Kirkpatrick and Dahlquist (2006) define *technical analysis* as the study of prices in freely traded markets with the intent of making profitable trading or investment decisions,[17] hence the models will only use daily prices for the selected stocks, ignoring company financial data for both forecasting methods, ANNs and autoregressive models.

1.1 Efficiency in the market

The Efficient Market Hypothesis states that stock prices reflect all available information and respond to any changes in that information through price changes. An implication of this relationship between available information and the market is that consistently predicting a change in an asset price is hard. Followers of this hypothesis believe that the market instantly responds to new information made available and so technical analysis of assets is moot. The Efficient Market Hypothesis finds its roots in the works of Bachelier (1900), Mandelbrot (1963) and Samuelson (1965) but more closely associated to Fama whose published review paper, *'Efficient Capital Markets: A Review of Theory and Empirical Work'*, pro-

posed the market efficiency types and the *joint hypothesis problem* that introduces certain requirements for testing market efficiency.[2][21][27][9] Of course, the EMH being near or even completely untestable means that a lot of research has been done to provide insight to the problem of predicting the stock market, both for the hypethesis and against. Grossman and Stiglitz (1980) propose a model that suggests that the market is not perfectly efficient as information is costly and in an efficient market where asset prices fully reflect all available information, there would be no incentive for investors to acquire information, decreasing the efficiency of the market. Both Grossman and Stiglitz and Black agree on the requirement of investors with different levels of information available to them, suggesting that it makes markets inefficient.[12][3]

The *random walk hypothesis* is the financial concept that asset price fluctuations can be modelled as random walks, making them unpredictable. A random walk is a stochastic or random process that involves random changes of a value, in the context of finance it's most often described as the change in price, positive or negative, by a set step point.[7][8][5] If the *random walk hypothesis* and the closely related *efficient market hypothesis* hold true, it should be extremely hard to consistently model and attempt to forecast asset prices. The following sections however, will go through modern and traditional methods of stock market modelling and forecasting.

1.2 Outline of Paper

Sections 2 and 3 will provide a comprehensive review of relevant literature for both neural networks and autoregressive integrated moving average models. Section 4 contains descriptions of the ARIMA models and the LSTM model, including optimisation techniques. Section 4 ends with a description of the evaluation metrics chosen for the comparison. Section 5 begins with a brief description of the preprocessing performed to the data, followed by the results of the project and a comparison between the two models, showing the LSTM models outperform the traditional ARIMA models. Section 6 will offer discussion on the results of the evaluation section for both models regarding time-series forecasting. The appendix contains the totality of graphs and values extracted from the training and testing of the models.

2 Overview of Neural Networks

In this section, we provide a brief history of neural networks in the context of finance and time series prediction, as well as a more detailed description of the architecture used in the paper.

Neural networks are systems largely belonging to the study of Machine Learning, typically associated with solving computational problems that other models and algorithms struggle to. Loosely based on biological neural networks, like the brain, artificial neural networks are collections of neuron-like nodes and links that connect them. Their resemblance to the brain in terms of architecture is part of what allowed neural networks to grow in computing power and popularity over the past decades, from their emergence in the 1940s.[18][22]

2.1 Neurons

Neurons are the artificial equivalent of a biological neuron and are the fundamental components of a neural network. Neurons perform three tasks, receive input from other neurons, apply an activation function to the input and output a value to other neurons. Neurons in the network are connected and each of these connections carries a signal and has certain weight attached to it, which affects the signal carried.[29][26] Graphically, a neuron could be represented in figure 2.1.

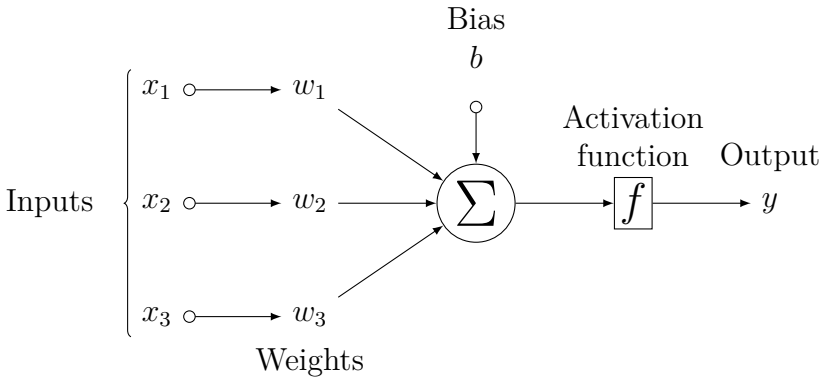


Figure 2.1: Graphical representation of a Neuron

2.2 Activation Functions

Wilson (2009) defines the activation function of a neural network as the function that governs the output behaviour of a neuron, given a set of input values. There are several types of activation functions, the simplest being the step function.

In mathematical terms, the step function, or the unit step function, is defined as

$$H(x) := \begin{cases} 0, & \text{for } x < 0 \\ 1, & \text{for } x \geq 0 \end{cases} \tag{2.1}$$

where $H(x)$ is the Heaviside step function.[1] At value 0, an output of $H(0) = 1$ is selected and passed down to the next layer of neurons.

2.2.1 Rectified Linear Unit

The Rectified Linear Unit (ReLU) is an activation function mathematically defined as

$$f(x) = [x]^+ = \max(0, x) \tag{2.2}$$

where x is the neuron's input value. Plotted on cartesian axes, the graph shows a linear relationship for $f(x)$ and x where $x > 0$. Generating a plot of ReLU further explains its activation range.

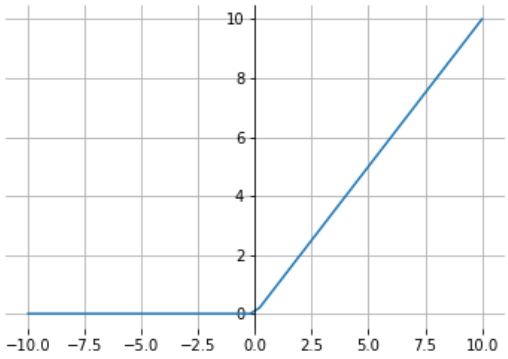


Figure 2.2: Plot of the Rectified Linear Unit function, generated using Python

ReLU was first introduced in the 2000 and 2001 Hahnloser papers.[13][14] ReLU is currently the most widely used activation function[24] and has found great success in training deep neural networks primarily in the fields of computer vision[10] and speech recognition.[20]

2.2.2 Sigmoid

The Sigmoid function is a logistic activation function, mathematically defined as

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}} \tag{2.3}$$

where

L is the maximum point of the curve,

k is the steepness of the curve,

x_0 is the value of the midpoint of the curve.

Sigmoid functions are non-linear, differentiable, defined for all real input values and have an output greater than zero for all inputs, in contrast to the ReLU activation function whose output is greater than zero only with positive inputs. A well known example of a sigmoidal curve is the cumulative distribution function of the normal distribution.

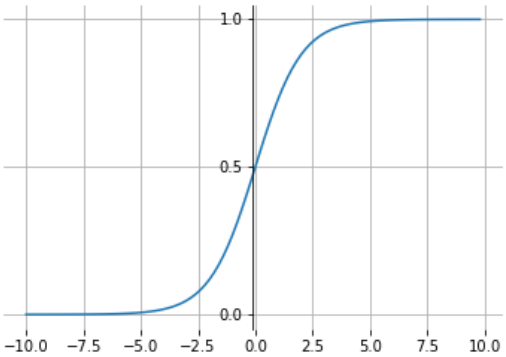


Figure 2.3: Plot of the sigmoid function, generated using Python

2.2.3 Hyperbolic tangent

The hyperbolic tangent is a specialised case of the sigmoid function, mathematically defined as

$$f(x) = \tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{2.4}$$

Unlike the standard sigmoid function, tanh can output negative values for negative inputs giving it double the range that the standard sigmoid has.

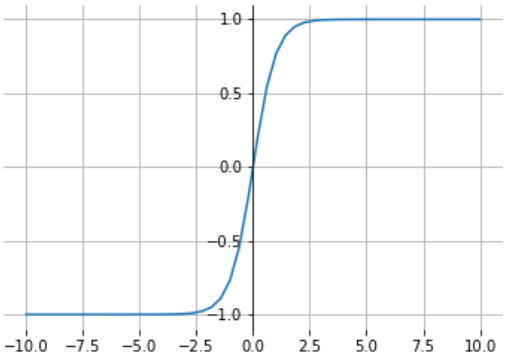


Figure 2.4: Plot of the hyperbolic tangent function, generated using Python

2.3 Backpropagation

The backpropagation algorithm is a key feature of neural networks as it allowed for fast and efficient training of multi-layered networks by distributing error values back through the layers of the network, adjusting the weights in the connections between neurons respectively.[28] In more detail, backpropagation functions compute the gradient $\nabla_x f(\mathbf{X}, \mathbf{y})$ numerically in a simple and inexpensive procedure.[25] What computing the gradient really means is calculating the derivative of the loss function in the model, with the loss function being a selected error function, a historically popular function being the sigmoid function described in the previous section.

The chain rule is a way to calculate the derivate of functions by decomposing a function to other functions whose derivative is known. Let x be a real number and f and g be functions that map from $\mathbb{R}$ to $\mathbb{R}$. By generalising the chain rule of calculus

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx} \tag{2.5}$$

one could compute the partial derivatives of the error at node (i, j) with weight w

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i} \tag{2.6}$$

Computing all these derivatives and putting them in a matrix of size $n \times m$ where $n = i$ and $m = j$ constructs the Jacobian matrix for the function $z = f(x)$. A Jacobian matrix is formally defined as the matrix of all first-order partial derivatives of a vector-valued function.

$$\mathbf{J} = \begin{bmatrix} \dfrac{\partial \mathbf{f}}{\partial x_1} & \cdots & \dfrac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_m}{\partial x_1} & \cdots & \dfrac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Figure 2.5: Matrix representation of the Jacobian matrix[16] $\mathbf{J}$ of function $\mathbf{f}$. Every entry (i, j) in the matrix is $\mathbf{J}_{i,j} = \frac{\partial f_i}{\partial x_j}$

Backpropagation performs this calculation for every single node in the network moving backwards, updating the weights on the nodes as it moves, effectively training the network.[11] Backpropagation is so efficient, it has become the most popular algorithm in training neural networks, along with the sigmoid being one of the most popular activation functions demonstrated in backprop algorithms due to its easy and simple derivative.

2.4 LSTM

Long Short-Term Memory (LSTM) is a type of recurrent neural network first introduced in 1997 by Hochreiter and Schmidhuber. Like all recurrent networks, LSTM differs from feed-forward networks by incorporating cyclical connections between neurons.

The LSTM was introduced specifically to address the two problems of the *vanishing gradient problem* and the *exploding gradient problem*. The two problems are closely linked problems regarding the propagation of errors in deep neural networks. In short, the two gradient problems appear when attempting to train networks with gradient-based learning, iterative methods for computing the gradient at each sample point, like the backpropagation method

described above, and either cause the gradient to be "vanishingly" small (*vanishing gradient*) or to "explosively" increase (*exploding gradient*).[30][31]

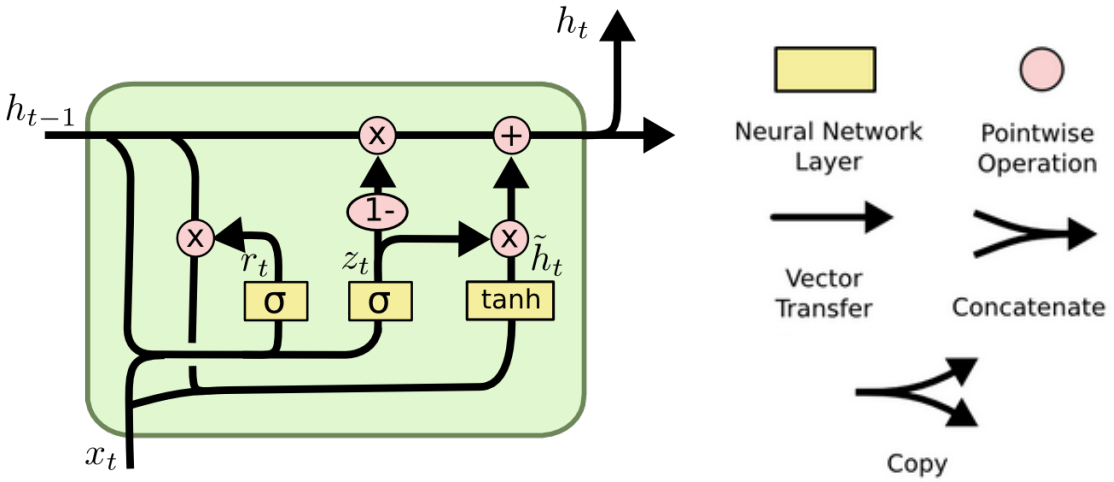


Figure 2.6: Diagram of the LSTM cell[23]

LSTM introduces a few extra units to deal with these problems, 3 additional gating units that control the state of the LSTM cell. In the original LSTM paper, Hochreiter and Schmidhuber used the sigmoid function as well as the related hyperbolic tangent function with varying ranges to control the state of the cells, leading to better performance in "remembering" long-term information about the structure of the data.[15]

A first sigmoid input gate affects the way old data are stored in the cell state,

$$f_t = \sigma\left(W_f[h_{t-1}, x_t] + b_f\right) \tag{2.7}$$

followed by a tanh gate updating the state with values from the current input.

$$i_t = \sigma\left(W_i[h_{t-1}, x_t] + b_i\right) \tag{2.8}$$

$$\hat{C}_t = tanh\left(W_C[h_{t-1}, x_t] + b_C\right) \tag{2.9}$$

The state cell is then updated and output is produced with yet another sigmoid function

$$C_t = f_t * C_{t-1} + i_t * \hat{C}_t \tag{2.10}$$

$$o_t = \sigma\left(W_o[h_t - 1, x_t] + b_o\right) \tag{2.11}$$

$$h_t = o_t * tanh(C_t) \tag{2.12}$$

3 Autoregressive models

Time series are series of points sorted by time. In the context of finance, time series are discrete-time data sequences, describing the nature of a financial instrument, with an example being the daily closing price of a stock. Mathematically, a time series X is written

$$X = \{X_1, X_2, ..., X_t\} \tag{3.1}$$

or alternatively

$$X = \{X_t : t \in T\} \tag{3.2}$$

3.1 Stochastic Processes

Stochastic processes are processes used to describe the behaviour of random variables and were first described by Luis Bachelier in 1900. Bachelier used discrete-time random walks to generate Brownian motion in his novel model for valuing stock options, making this the first instance in finance to use advanced mathematics to model instruments.[2] Box and Jenkins define stochastic processes as models that describe the probability structure of a sequence of observations.[4] For stochastic process to be useful in forecasting values, they need to be of a specific form and vary in stable, predictible manners. A stationary process is a subclass of stochastic processes whose statistical properties, e.g. the mean or the variance, do not change with time.[19] Stationarity is a key concept in time series analysis, with many theorems directly assuming stationarity in the data. The simplest example of a stationary stochastic process is the example of white noise, $X_t = \epsilon_t$.

The ARMA model is a description of a stochastic process as a combination of two stationary polynomial terms, AR and MA, where

AR $\rightarrow$ Autoregression,

MA $\rightarrow$ Moving Average.

Autoregressive models of order p are denoted as AR(p) and are mathematically defined as

$$X_t = c + \sum_{i=1}^{p} \varphi_i L^i X_t + \epsilon_t \tag{3.3}$$

where

φ_i are the parameters of the model,

L^i is the Lag operator (also known as BackShift operator).

The Lag operator L is defined for $X = \{X_1, X_2, ..., X_t\}$ as

$$LX_t = X_{t-1} \tag{3.4}$$

$$L^k X_t = X_{t-k} \tag{3.5}$$

Moving Average models of order q are denoted as MA(q) and are mathematically defined as

$$X_t = \mu + \epsilon_t + \sum_{i=1}^{q} \theta_i L^i \epsilon_t \tag{3.6}$$

where

θ_i are the parameters of the model.

Put together, these terms create the ARMA model, denoted as ARMA(p, q) and mathematically defined as

$$X_t = c + \epsilon_t + \sum_{i=1}^{p} \varphi_i L^i X_t + \sum_{i=1}^{q} \theta_i L^i \epsilon_t \tag{3.7}$$

ARIMA is a generalisation of the ARMA model applied to non-stationary processes. *I* stands for *Integrated* and refers to the differencing performed to the data to make them stationary.

By moving ARMA terms around we show that

$$\left(1 - \sum_{i=1}^{p} \varphi_i L^i\right) X_t = c + \left(1 + \sum_{i=1}^{q} \theta_i L^i\right) \epsilon_t \tag{3.8}$$

Reducing further to

$$\varphi_p(L) X_t = \theta_q(L) \epsilon_t \tag{3.9}$$

where

$\varphi_p(L) = 1 - \sum_{i=1}^{p} \varphi_i L^i$

$\theta_q(L) = 1 + \sum_{i=1}^{q} \theta_i L^i$

When our process exhibits non-stationarity, differencing is performed to make the process stationary.

$$\begin{aligned} Y_t &= X_t - X_{t-1} = (1-L)X_t \\ Y_t - Y_{t-1} &= X_t - 2X_{t-1} + X_{t-2} \\ &= (1-L)^2 X_t \end{aligned} \tag{3.10}$$

Generalising

$$Y_t - \sum_{i=1}^{d} Y_{t-k} = (1-L)^d X_t \tag{3.11}$$

Introducing this general differencing operator to the ARMA model transforms it into an ARIMA(p, d, q) model

$$\varphi_p(L)(1-L)^d X_t = \theta_q(L)\epsilon_t \tag{3.12}$$

where

p is the order of autoregressive model,

d is the order of differencing,

q is the order of moving average model.

3.2 Model Selection

A popular methodology for selecting ARIMA models to fit data to is the Box-Jenkins methodology, first presented in 1970 by statisticians George Box and Gwilym Jenkins in their textbook 'Time series analysis: Forecasting and control.' The model is used to determine the order of ARIMA model to be used, in terms of p, d and q, as defined above. There are three steps to the model: Model Identification, Model Estimation and Diagnostic Checking. Model Identification refers to using plots of the data as long as other information about the structure of the time series to "guess" the class of ARIMA model to be used. During this step, the autocorrelation and partial autocorrelation functions are plotted and reasonable guesses can be made about the order of the model. Model Estimation refers to using numerical methods to approximate the true values of p, d and q and Diagnostic Checking refers to checking the model after it has been fitted to the data by plotting autocorrelation plots for the residual terms.

4 Methodology

The following section provides details in the construction of the model for predicting stock prices, as well as a description of the ARIMA model to which we compare the network.

4.1 ARIMA

To fit an ARIMA model we need to first perform some basic data exploration to identify key features in the time series. Our aim is to attempt to identify the order of the model through observations of the plots instead of relying on innefficient but exhaustive grid search.

In total, there are three model parameters we need to identify:

- p, autoregressive terms
- d, integrated terms
- q, moving average terms

We begin by plotting autocorrelation functions for the entire dataset and each stock individually. Slow degradation of the ACF plot tends to point to autoregressive terms while the opposite is true for moving average terms. Figure 4.1 shows us the autocorrelation plot for all the stocks. We observe a slow fall in the ACF plot, which is consistent with autoregressive terms in the model.

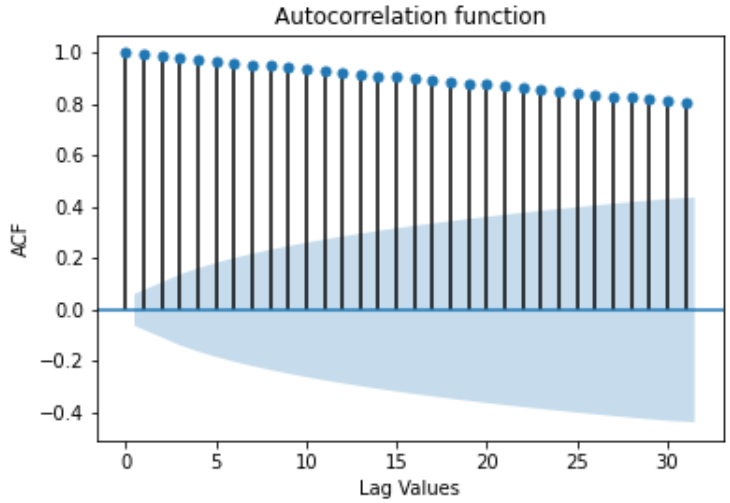


Figure 4.1: Autocorrelation function before any differencing, generated using Python

This graph also tells us that there is significant correlation in the series for over 30 lags. No differencing has been done at this point, the strong correlation could be caused by auto-correlation at lags 1 or 2, which is confirmed by the Partial Autocorrelation function plot below:

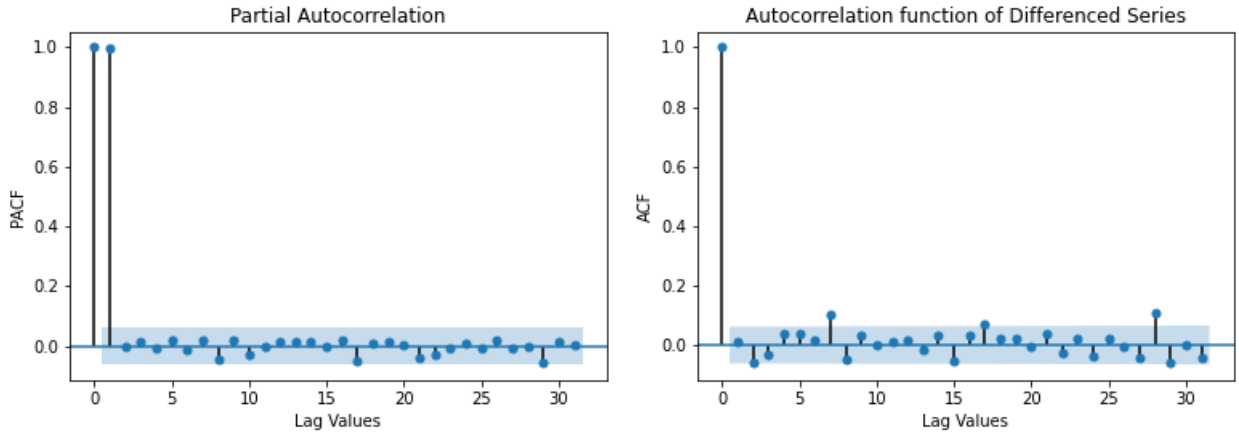


Figure 4.2: PACF and ACF of differenced series, generated using Python

We observe two significant spikes at lags 1 and 2, indicating that the series needs to be differenced twice, $d = 2$. We also observe a major positive spike at $lag = 1$ for the differenced ACF plot on the right, which suggests we should add another autoregressive term.

We conclude that a good estimate on the hyperparameters of the ARIMA model we are going to use would be (2, 2, 0), for 2 autoregressive terms, 2 differencing terms and 0 moving average terms.

Using the `statsmodel` Python module, we create and validate our ARIMA model using walk-forward validation:

```
    train, test = X[:train_size], X[train_size:]
    predictions = []
    for t, _ in enumerate(test):
        model = ARIMA(train, order=(2,2,0))
        model_fit = model.fit(disp=False)
        yhat = model_fit.forecast()[0]
        predictions.append(yhat)
        train.append(test[t])

    mse = mean_squared_error(test, predictions)
    mae = mean_absolute_error(test, predictions)
```

Figure 4.3: Walk Forward Validation of ARIMA model using the `statsmodels` library

A training and testing split of 80-20 was used to evaluate ARIMA, the same split we use to evaluate the neural network approach.

4.2 LSTM

The network is a relatively simple network by most accounts, comprised of a single hidden layer. The simplicity of the model only goes to show the power of neural networks in fitting and forecasting time-series. The model consists of three layers, an input layer, a Long Short-Term Memory (LSTM) layer and the output layer, as shown in 4.4.

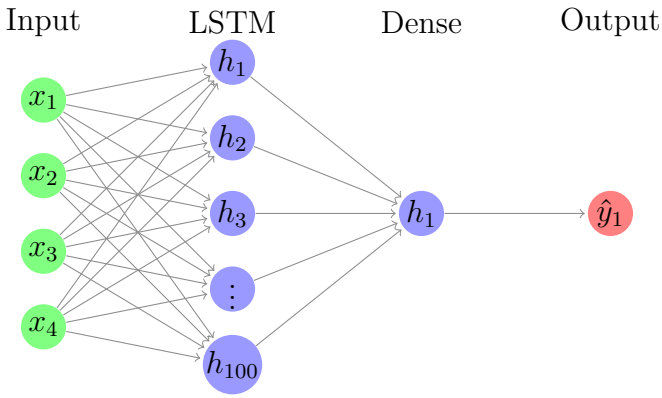


Figure 4.4: Model Architecture, generated using LaTeX

The input layer receives a 2-dimensional array of historical stock values, including the previous day's opening, highest, lowest and closing stock prices. The network then allows the neurons to compete amongst each other and determing an appropriate output. Output is in the shape of a 1-dimensional array containing four values, the future day's predicted stock values for open, high, low and close. The LSTM layer contains 100 nodes and is trained for 100 epochs. The Rectifier Linear Unit was used as its activation function and the Mean Absolute Error was used as its loss function.

Nodes	**Epochs**	**Optimizer**	**Learning Rate**	**Activation**	**Loss**
100	100	Adam	0.001	ReLU	MAE

Table 4.1: Description of the hidden LSTM layer

In this project we made use of the Keras API to implement the network. Keras is written in Python and operates ontop of TensorFlow, which is an extremely popular and widely used Deep Learning library.[6] A simple sequential model using LSTM cells built using Keras would look like this:

```
1  model = Sequential()
2  model.add(LSTM(100, activation="relu", input_shape=(n_steps, n_features)))
3  model.add(Dense(n_features))
4  opt = Adam(learning_rate=0.001)
5  model.compile(optimizer=opt, loss="mae", metrics=["mse"])
```

With as little as 5 lines of code, we have a working Long Short-Term Memory model ready for training. The LSTM cell easily remembers the long term dependencies in the data and outputs a 1-dimensional array containing future values. Figure 4.5 contains a more detailed breakdown of the layers used in the network.

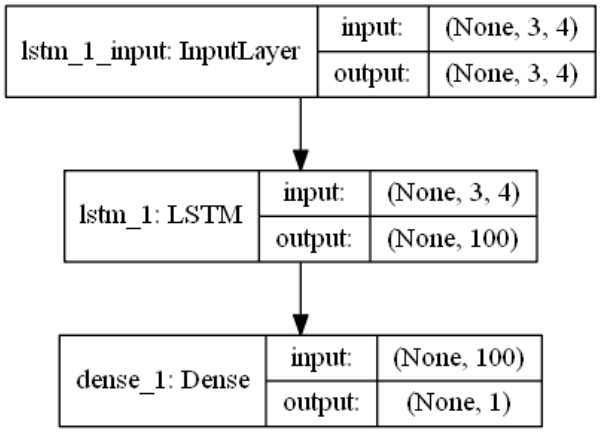


Figure 4.5: Description of layers, generated using Python

A Dense layer is a fully-connected feedforward layer. Feedforward layers are the simplest type of architecture employed in machine learning, which is essentially a collection of neurons that pass signals in one direction only, the layer in front of them. These layers do not form cycles in their connections which is fundamentally different from Recurrent Neural Networks like the LSTM. The purpose of the Dense layer in the model is to reduce the dimensionality of the data from the 100-dimensional space created in the LSTM down to a 1-dimensional array of values, which is then lead to the output layer. The Dense layer performs no operations on the data, it's activation function is linear.

4.3 Model Evaluation

To perform an evaluation of the model, we feed into it the testing set gathered from the dataset. The model uses the Mean Absolute Error metric to guide the minimisation of its loss function during training. Mathematically, the Mean Absolute Error is defined as

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i| \tag{4.1}$$

and is the sum of the absolute differences of predicted data points to actual data points divided by the number of data points in the set. MAE is also used in the evaluation of the compiled and trained model.

Evaluation of the model also includes a comparison of Mean Squared Error values. MSE is mathematically defined as

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \tag{4.2}$$

and is the mean of the squared differences between the predicted values and the actual data points. MAE was chosen for the training of the model because of its linear relationship between penalties and errors, in that it treats a predicted to actual difference of 1 in a way proportionally to how it treats a predicted to actual difference of 5. MSE treats larger differences between predictions and actual data non-linearly.

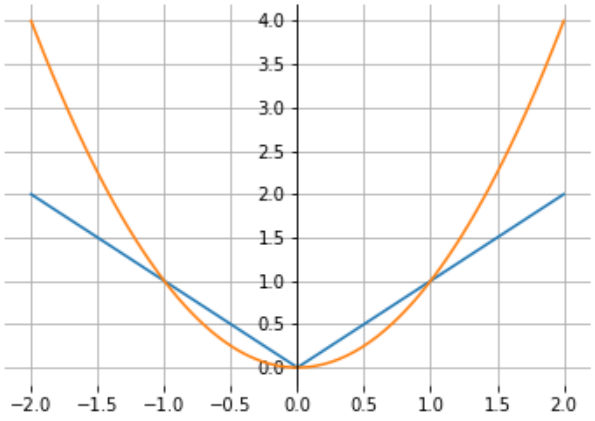


Figure 4.6: Comparison of MAE (blue) and MSE (orange). Note how MAE scales linearly with higher error values while MSE scales quadratically, treating higher errors more harshly.

5 Results

5.1 Dataset

We retrieve data on tickers available at `https://www.tiingo.com/`. The stock exchanges targeted in this project are the New York Stock Exchange (NYSE) and the National Association of Securities Dealers Automated Quotations (NASDAQ). The data comprise of companies from the technology and the financial services industries.

The data consist of daily values for the period starting January 1, 2016 and ending December 31, 2019. The dataset includes daily information for the `close`, `open`, `high` and `low` prices of each trading day. Below is a table listing all stocks considered for the project:

Stock Name	**Symbol**	**Stock Exchange**
Apple Inc.	AAPL	NASDAQ
Amazon Inc.	AMZN	NASDAQ
American Express Company	AXP	NYSE
Boeing Co	BA	NYSE
Bank of America Corp	BAC	NYSE
Citigroup Inc	C	NYSE
Ford Motor Company	F	NYSE
Facebook Inc.	FB	NASDAQ
General Electric Company	GE	NYSE
Alphabet Inc. Class C	GOOG	NASDAQ
Goldman Sachs Group Inc.	GS	NYSE
JPMorgan Chase & Co.	JPM	NYSE
Morgan Stanley	MS	NYSE
Microsoft Corporation	MSFT	NASDAQ
Wells Fargo & Co.	WFC	NYSE

Table 5.1: Stocks included in the study

The data range wildly from low closing values of a few dollars to thousands of dollars per stock. Before the data is given to the model, we normalise to the range of [0, 1]. The data range of Jan 1, 2016 - Dec 31, 2019 includes 1006 rows of daily stock data from which we select the 1000 most recent for modelling.

5.2 Preprocessing

To normalise the data, we use `scikit-learn`'s `MinMaxScaler` which scales features to the range [0, 1] based on the minimum and maximum value in the set. We select all the features we are going to use in the model, `open`, `high`, `low` and `close` and stack them horizontally in a single multidimensional array to be scaled down.

```
1       open_ = dataset["open"].values.reshape(1000, 1)
2       high = dataset["high"].values.reshape(1000, 1)
3       low = dataset["low"].values.reshape(1000, 1)
4       close = dataset["close"].values.reshape(1000, 1)
5       d = np.hstack((open_, high, low, close))
6       scaler = MinMaxScaler()
7       d = scaler.fit_transform(d)
```

Figure 5.1: Simple normalisation of data using `MinMaxScaler` from the `sklearn` library

The transformation of the data is given by the mathematical formula:

$$X'_{i,j} = \frac{X_{i,j} - X_{min}}{X_{max} - X_{min}} \tag{5.1}$$

And the inverse transform is given by the mathematical formula:

$$X_{i,j} = X'_{i,j}(X_{max} - X_{min}) + X_{min} \tag{5.2}$$

where

i is the trading day,

j is the column number, indicating whether the value belongs to `open`, `high`, `low` or `close` prices

Labels are then extracted from the normalised dataset. We attempt to predict the closing price for the next day, given a few values, the current trading day's stock values. We split the data in such a way that for each trading day, the inputs (or features) are arrays of the `open`, `high`, `low` and `close` values of the previous specified number of days and the output (or labels) are an array of the predicted `open`, `high`, `low` or `close` values.

An example to help visualise the data split:

With an input of:

```
array([[656.29 , 657.715, 627.51 , 636.99 ],
       [646.86 , 646.91 , 627.65 , 633.79 ],
       [622.   , 639.79 , 620.31 , 632.65 ],
       [621.8  , 630.   , 605.21 , 607.94 ]])
```

and a step size of 3 days, we split the data into:

```
X = array([[656.29 , 657.715, 627.51 , 636.99 ],
           [646.86 , 646.91 , 627.65 , 633.79 ],
           [622.   , 639.79 , 620.31 , 632.65 ]])

y = array([621.8 , 630.  , 605.21, 607.94])
```

Figure 5.2: Splitting of data into features and labels

where

`X` are stock values from the past 3 trading days, given to the model as input (features),
`y` are desired output values for all stock values we wish to predict as output (labels)

The data set is then split up into training and testing sets. The LSTM model is trained on the training set and then evaluated on the testing set. The ARIMA model is fitted and evaluated using Walk-Forward evaluation and MSE and MAE values are generated for both models and for all available stocks.

5.3 Evaluation Metrics

For each of the stocks and for each of the models, two evaluation metrics are produced, the mean squared error (MSE) and the mean absolute error (MAE).

5.4 Results

Long Short-Term Memory outperforms the autoregressive model in every single stock prediction, with lower error values for both error metrics, MAE and MSE.

	LSTM		**ARIMA**	
Ticker	**MAE**	**MSE**	**MAE**	**MSE**
AAPL	0.0118077	$2.45 \cdot 10^{-4}$	0.0144261	$3.427 \cdot 10^{-4}$
AMZN	0.0116115	$2.312 \cdot 10^{-4}$	0.0129651	$2.859 \cdot 10^{-4}$
AXP	0.0136267	$3.001 \cdot 10^{-4}$	0.0159697	$3.948 \cdot 10^{-4}$
BA	0.0151952	$4.118 \cdot 10^{-4}$	0.0176966	$4.833 \cdot 10^{-4}$
BAC	0.0114324	$2.433 \cdot 10^{-4}$	0.0150852	$3.816 \cdot 10^{-4}$
C	0.0148086	$3.893 \cdot 10^{-4}$	0.0214394	$7.23 \cdot 10^{-4}$
F	0.0146274	$4.574 \cdot 10^{-4}$	0.0200318	$7.796 \cdot 10^{-4}$
FB	0.0151909	$4.407 \cdot 10^{-4}$	0.0218248	$7.753 \cdot 10^{-4}$
GE	0.0060934	$7.28 \cdot 10^{-5}$	0.0066543	$8.51 \cdot 10^{-5}$
GOOG	0.0149529	$5.539 \cdot 10^{-4}$	0.0210107	$9.553 \cdot 10^{-4}$
GS	0.0157465	$4.234 \cdot 10^{-4}$	0.0200464	$6.25 \cdot 10^{-4}$
JPM	0.0117033	$2.4 \cdot 10^{-4}$	0.0143821	$3.343 \cdot 10^{-4}$
MS	0.0121079	$2.525 \cdot 10^{-4}$	0.0150503	$3.616 \cdot 10^{-4}$
MSFT	0.0106946	$2.09 \cdot 10^{-4}$	0.0133162	$2.835 \cdot 10^{-4}$
WFC	0.0169028	$5.05 \cdot 10^{-4}$	0.0258048	$1.0333 \cdot 10^{-3}$

Table 5.2: MAE and MSE results for both models

6 Discussion

Averaging the values for the two models across all stocks further shows the improved performance of the LSTM compared to the traditional ARIMA model.

Model	**MAE**	**MSE**
LSTM	0.013100	0.000332
ARIMA	0.017047	0.000523

Table 6.1: Average MAE and MSE results

This doesn't mean that ARIMA and other statistical models are to be discarded in favour of neural networks. Neural networks are still relatively new to the world and are still being improved upon. This paper was an attempt to show how easy and simple it is to set up a recurrent neural network without much optimisation that would outperform more established statistical models, but it is not a comprehensive comparison of the two models. Granted, the ARIMA model was not optimised as much as it could have been with only elementary observations guiding the model selection stages instead of something more robust like an exhaustive hyperparameter grid search.

This paper used a single hidden LSTM layer with 100 neurons at 100 epochs to achieve an average mean absolute error of 0.0131, approximately 23% lower than the ARIMA model. Further research could improve upon the model selection and hyperparameter estimation for both models, with testing of other ARIMA orders and exploring more options for the LSTM model, like varying the number of epochs, the learning rate of the optimiser, altering the shape of the data, introducing more layers to the network and even creating new architectures to handle time series forecasting.

Future research could have other models be compared to LSTMs, like wavelet analysis, generalised autoregressive heteroscedastic models, as well as multivariate models.

References

[1] M. Abramowitz and I. A. Stegun, *Handbook of Mathematican Functions with Formulas, Graphs and Mathematical Tables*. National Bureau of Standards, 1972 (cit. on p. 4).

[2] L. Bachelier, 'The theory of speculation,' *Annales scientifiques de l'Ecole Normale Superieure*, vol. 3, no. 17, pp. 21–86, 1900 (cit. on pp. 1, 2, 9).

[3] F. Black, 'Noise,' *Journal of Finance*, vol. 41, no. 3, 1986. DOI: 10.1111/j.1540-6261.1986.tb04513.x (cit. on p. 2).

[4] G. Box and G. Jenkins, 'Time series analysis: Forecasting and control,' 1970 (cit. on pp. 9, 11).

[5] M. Burton, *A Random Walk Down Wall Street*. WW Norton & Company Inc, 1973 (cit. on p. 2).

[6] F. Chollet *et al.* (2015). 'Keras,' [Online]. Available: https://github.com/fchollet/keras (cit. on p. 15).

[7] P. Cootner, *The Random Character of Stock Market Prices*. MIT Press, 1964 (cit. on p. 2).

[8] E. Fama, 'Random walks in stock market prices,' *Financial Analysts Journal*, vol. 21, no. 5, pp. 55–59, 1965. DOI: 10.2469/faj.v21.n5.55 (cit. on p. 2).

[9] ——, 'Efficient capital markets: A review of theory and empirical work,' *Journal of Finance*, vol. 25, no. 2, pp. 383–417, 1970 (cit. on pp. 1, 2).

[10] X. Glorot, A. Bordes and Y. Bengio, 'Deep sparse rectifier neural networks,' *Artificial Intelligence and Statistics (AISTATS*, vol. 2011, 2011 (cit. on p. 5).

[11] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org (cit. on p. 7).

[12] S. J. Grossman and J. E. Stiglitz, 'On the impossibility of informationally efficient markets,' *American Economic Review*, vol. 70, no. 3, 1980 (cit. on p. 2).

[13] R. H. R. Hahnloser, R. Sarpeshkar and M. M. et al, 'Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit,' *Nature*, vol. 405, pp. 947–951, 2000. DOI: https://doi.org/10.1038/35016072 (cit. on p. 5).

[14] R. H. R. Hahnloser and H. S. Seung, 'Permitted and forbidden sets in symmetric threshold-linear networks,' *NIPS*, 2001 (cit. on p. 5).

[15] S. Hochreiter and J. Schmidhuber, 'Long short-term memory,' *Neural Computation*, vol. 9, no. 8, 1997 (cit. on pp. 7, 8).

[16] *Jacobian matrix*, https://en.wikipedia.org/wiki/Jacobian$_m atrix_a nd_d eterminant$, 2020 (cit. on p. 7).

[17] C. D. Kirkpatrick and J. R. Dahlquist, *Technical Analysis: The Complete Resource for Financial Market Technicians*. 2006, ISBN: 978-0-13-153113-0 (cit. on p. 1).

[18] S. Kleene, 'Representation of events in nerve nets and finite automata,' *Annals of Mathematics Studies*, pp. 3–41, 1956 (cit. on p. 3).

[19] P. Kun, *Fundamentals of Probability and Stochastic Processes with Applications to Communications*. Springer, 2018 (cit. on p. 9).

[20] A. L. Maas, A. Y. Hannun and A. Y. Ng, 'Rectifier nonlinearities improve neural network acoustic models,' *Stanford Press*, 2014 (cit. on p. 5).

[21] B. Mandelbrot, 'The variation of certain speculative prices,' *The Journal of Business*, vol. 36, no. 4, pp. 394–419, 1963. [Online]. Available: `http://www.jstor.org/stable/2350970` (cit. on pp. 1, 2).

[22] W. McCulloch and W. Pitts, 'A logical calculus of ideas immanent in nervous activity,' *A Logical Calculus of Ideas Immanent in Nervous Activity*, vol. 5, no. 4, 1943. DOI: `10.1007/BF02478259` (cit. on p. 3).

[23] C. Olan, *Understanding lstm networks*, http://colah.github.io/posts/2015-08-Understanding-LSTMs/, 2015 (cit. on p. 8).

[24] R. Prajit and B. Zoph, 'Searching for activation functions,' 2017 (cit. on p. 5).

[25] D. Rumelhart, G. Hinton and R. Williams, 'Learning representations by back-propagating errors,' *Nature*, vol. 323, pp. 533–536, 1986. DOI: `10.1038/323533a0` (cit. on p. 6).

[26] S. Russel and P. Norvig, *Artificial Intelligence, A Modern Approach*, Third. Pearson, 2010 (cit. on p. 3).

[27] P. A. Samuelson, *Proof that Properly Anticipated Prices Fluctuate Randomly*. 1965, ch. Chapter 2, pp. 25–38. DOI: `10.1142/9789814566926_0002` (cit. on pp. 1, 2).

[28] P. J. Werbos, *Beyond Regression: New Tools for Predictions and Analysis in the Behavioural Sciences*. Harvard University Press, 1975 (cit. on p. 6).

[29] W. H. Wilson, *The Machine Learning Dictionary*. Saint Elizabeth University, 2009 (cit. on pp. 3, 4).

[30] B. Y, S. P and F. P, 'Learning long-term dependencies with gradient descent is difficult,' *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994 (cit. on p. 8).

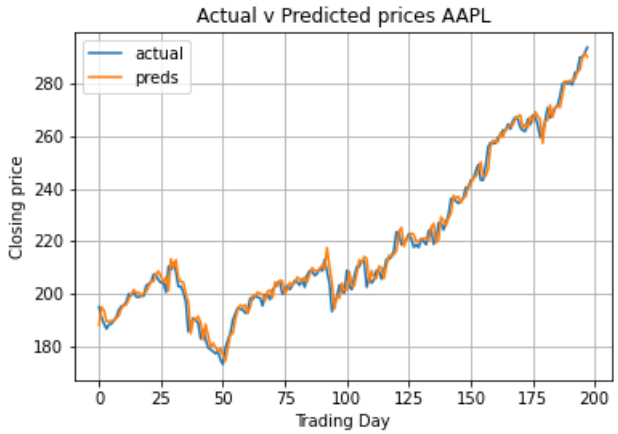
[31] B. Y, M. T and P. R, 'On the difficulty of training recurrent neural networks,' 2012. DOI: `1211.5063` (cit. on p. 8).

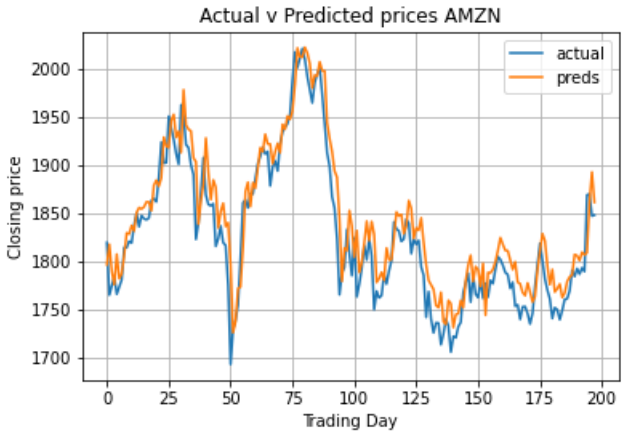
List of Tables

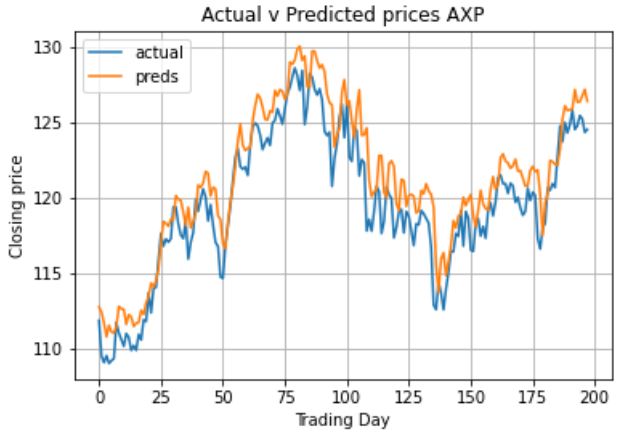
List of Figures

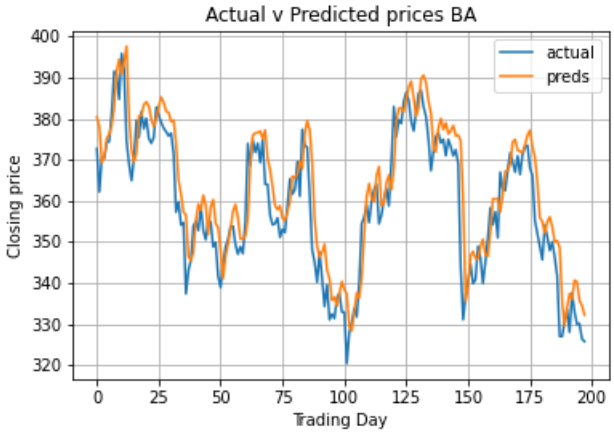
7 Appendix

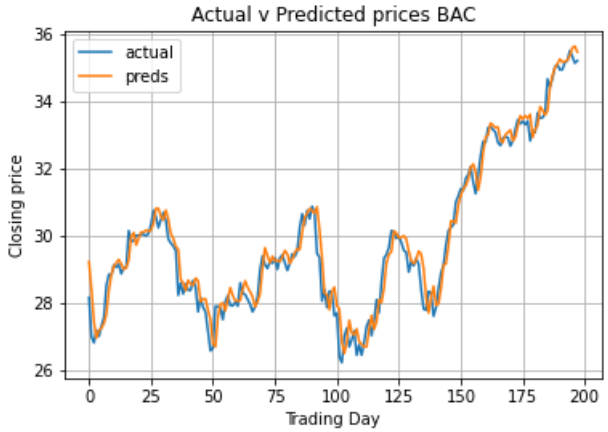
7.1 Prediction plots with LSTM

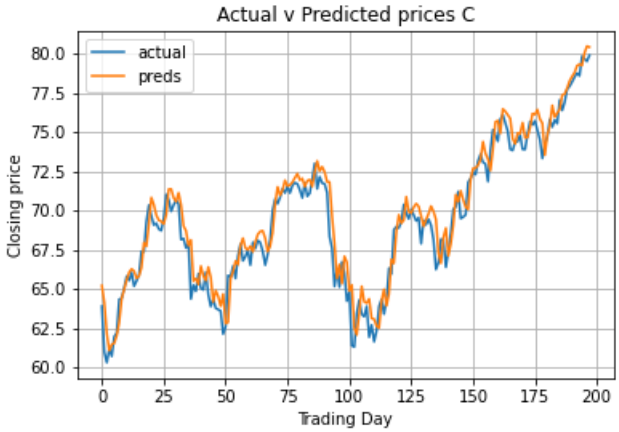


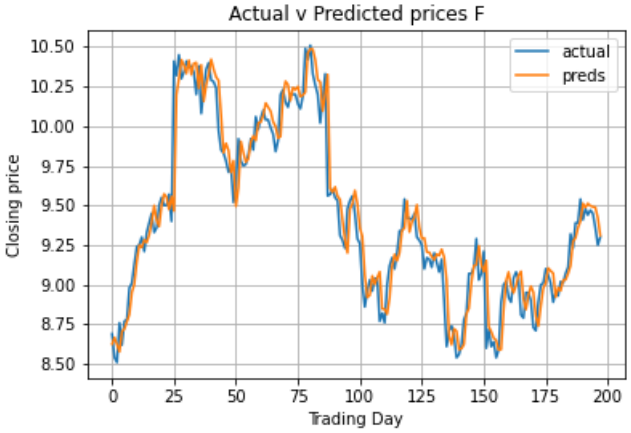


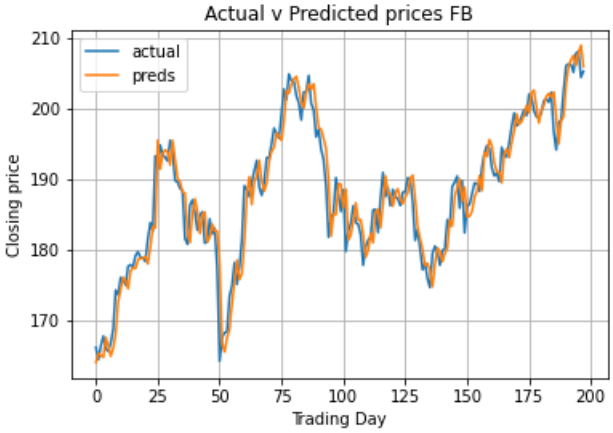


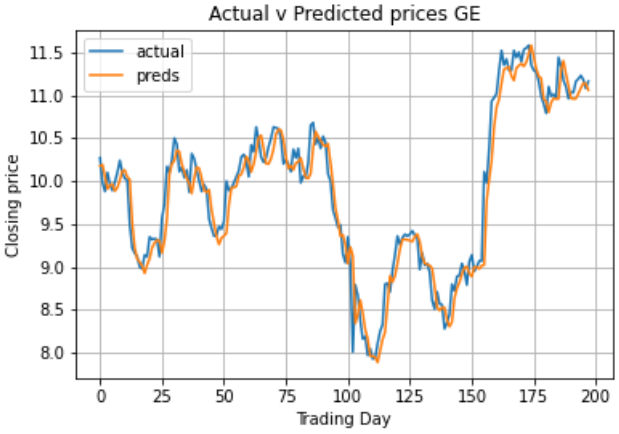


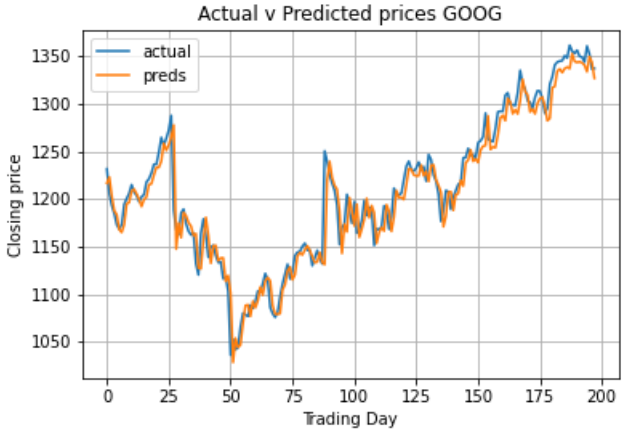


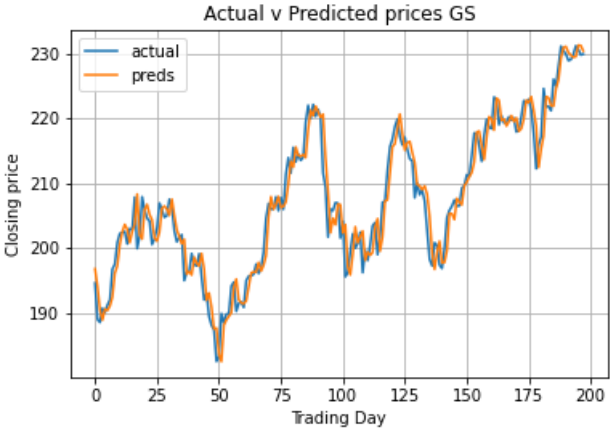


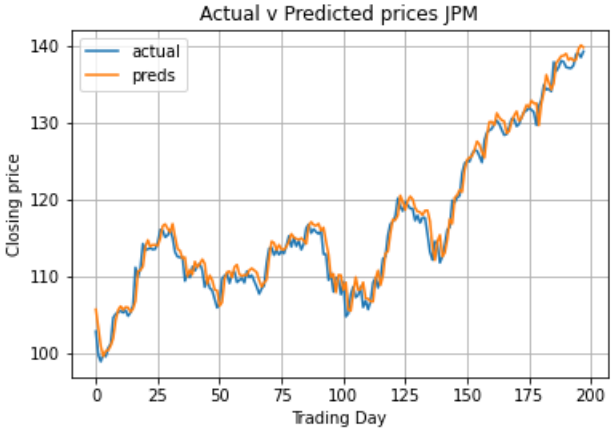

Actual v Predicted prices F
actual
preds
Closing price
Trading Day

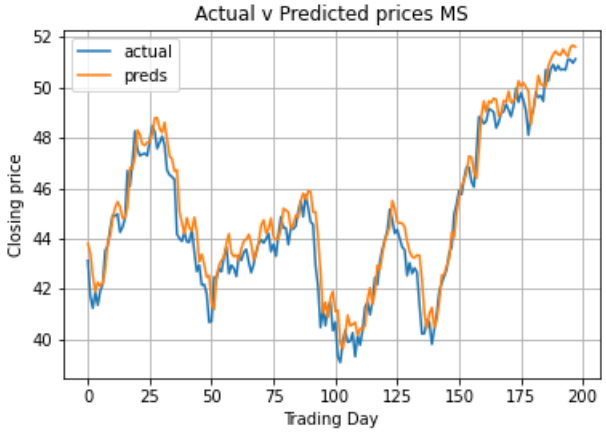

Actual v Predicted prices FB
actual
preds
Closing price
Trading Day


Actual v Predicted prices GE
actual
preds
Closing price
Trading Day

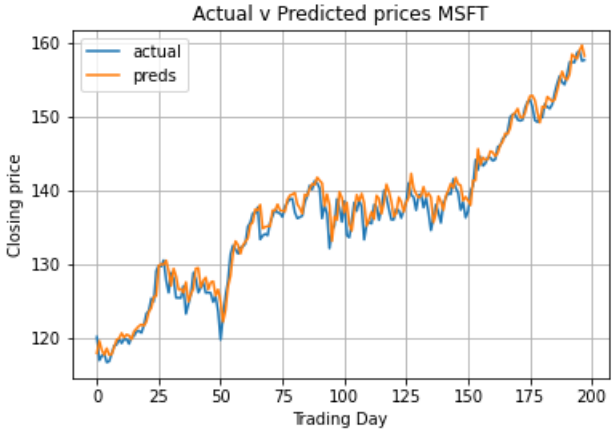

Actual v Predicted prices GOOG
actual
preds
Closing price
Trading Day

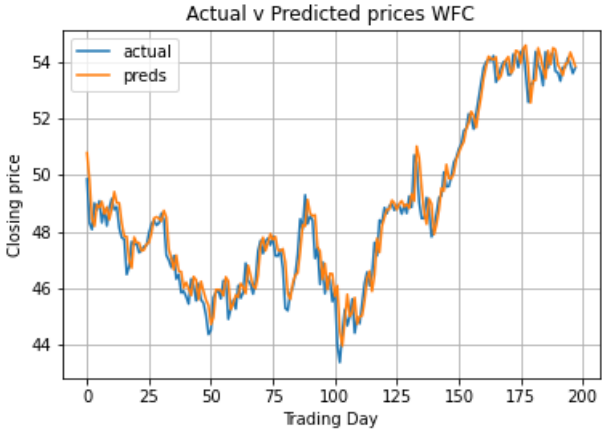

Actual v Predicted prices GS
actual
preds
Closing price
Trading Day


Actual v Predicted prices JPM
actual
preds
Closing price
Trading Day

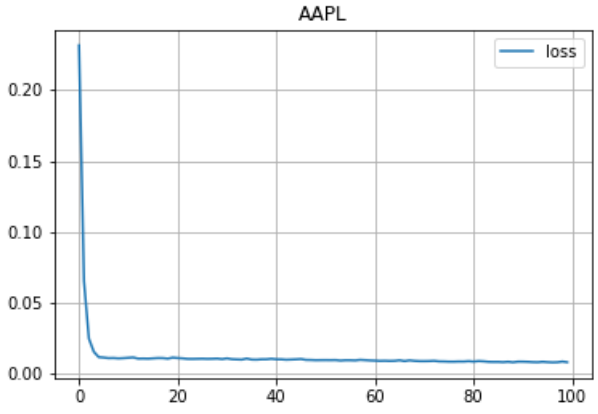


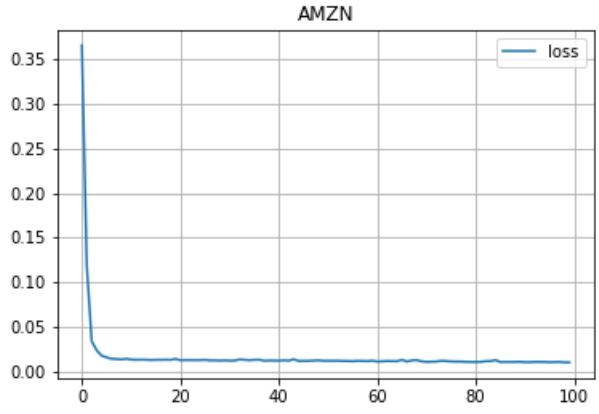
]

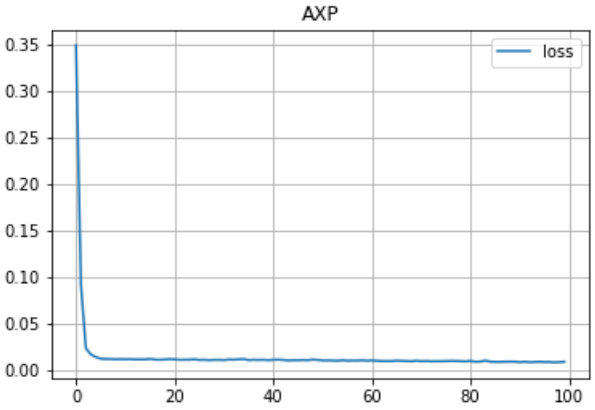


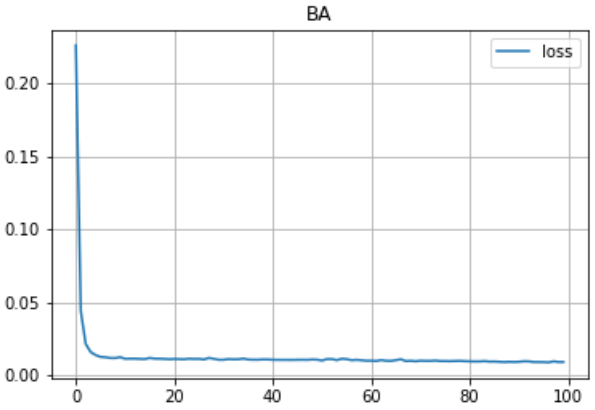


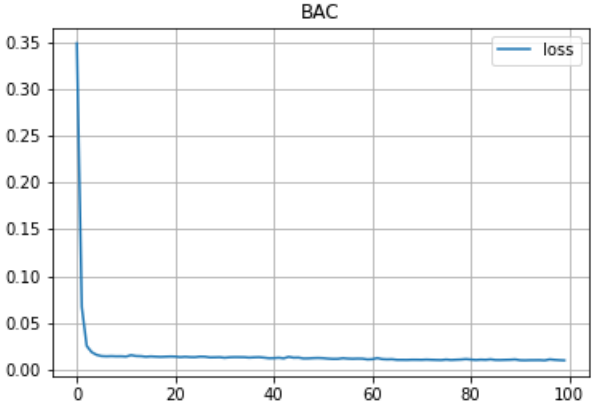
7.1 Training Loss functions



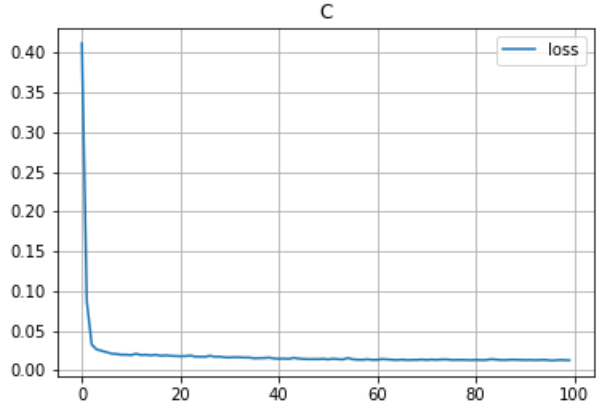



AXP
loss
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
20
40
60
80
100

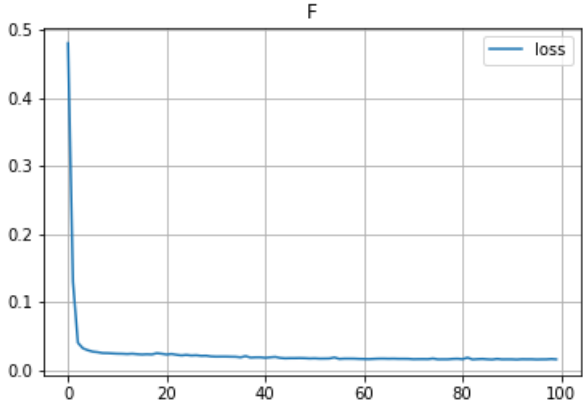

BA
loss
0.20
0.15
0.10
0.05
0.00
0
20
40
60
80
100

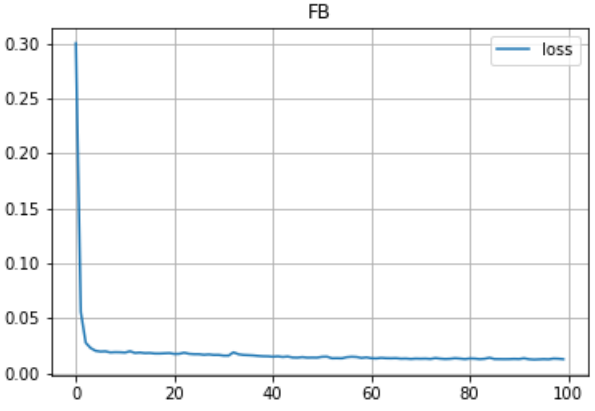

BAC
loss
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
20
40
60
80
100

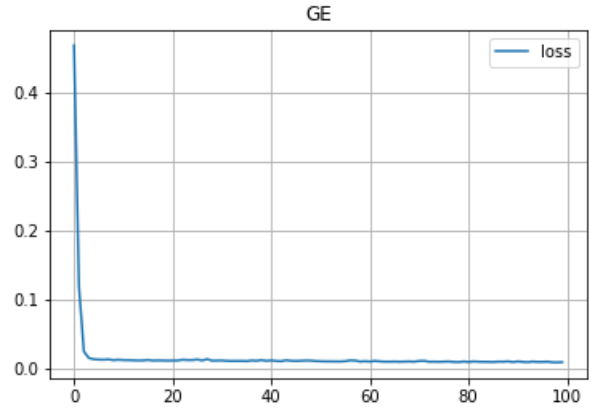
]

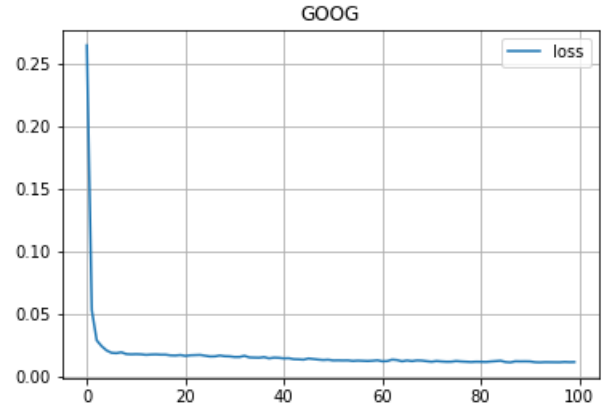

C
loss
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
20
40
60
80
100

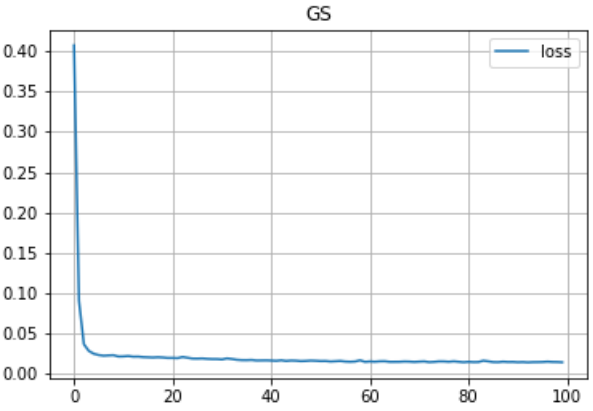
]

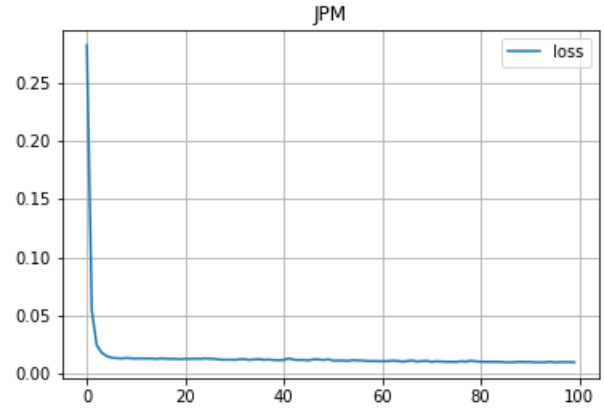

F
loss
0.5
0.4
0.3
0.2
0.1
0.0
0
20
40
60
80
100

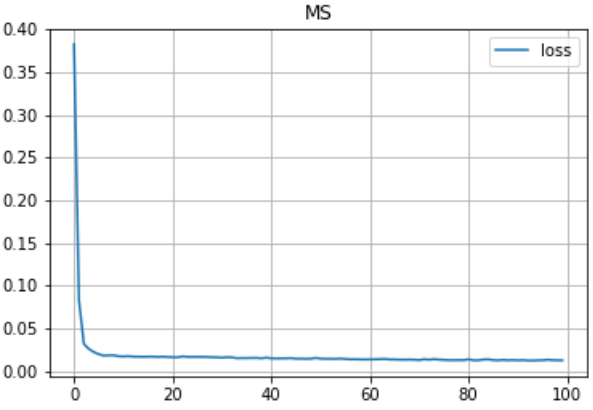

FB
loss
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
20
40
60
80
100

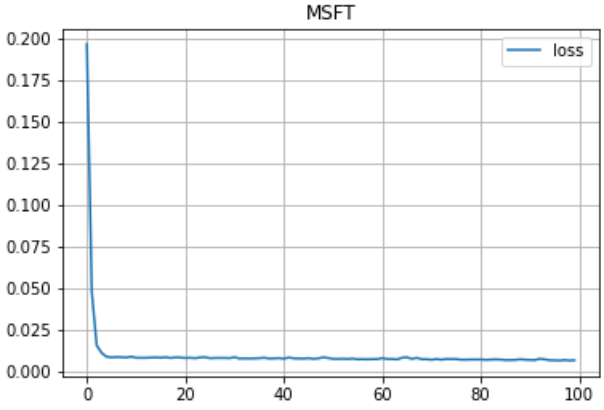

GE
loss
0.4
0.3
0.2
0.1
0.0
0
20
40
60
80
100

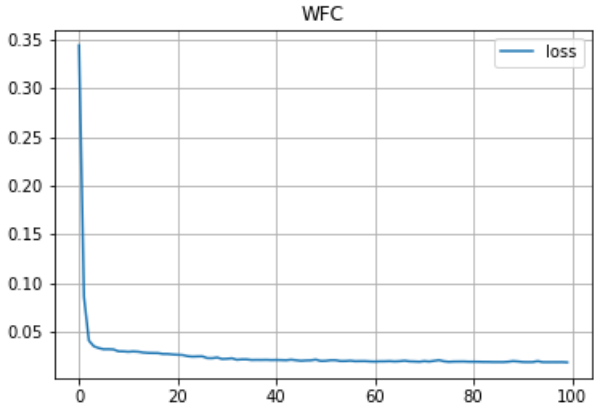

GOOG
loss
0.25
0.20
0.15
0.10
0.05
0.00
0
20
40
60
80
100


GS
loss
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
20
40
60
80
100


JPM
loss
0.25
0.20
0.15
0.10
0.05
0.00
0
20
40
60
80
100


MS
loss
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
20
40
60
80
100


MSFT
loss
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0
20
40
60
80
100


WFC
loss
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0
20
40
60
80
100