UNIVERSITY OF WARSAW | FACULTY OF ECONOMIC SCIENCES

WORKING PAPERS
No. 5/2023 (412)

THE PERFORMANCE OF TIME SERIES FORECASTING BASED ON CLASSICAL AND MACHINE LEARNING METHODS FOR S&P 500 INDEX

MAUDUD HASSAN UZZAL
ROBERT ŚLEPACZUK

WARSAW 2023

WORKING PAPERS 5/2023 (412)

The performance of time series forecasting based on classical and machine learning methods for S&P 500 index

Maudud Hassan Uzzal [a], *Robert Ślepaczuk* [b]

[a] *University of Warsaw, Faculty of Economic Sciences, Quantitative Finance Research Group,*
[b] *University of Warsaw, Faculty of Economic Sciences, Quantitative Finance Research Group, Department of Quantitative Finance*
Corresponding author: rslepaczuk@wne.uw.edu.pl

Abstract: Based on one step ahead forecasts, this study compares the forecasting abilities of the traditional technique (ARIMA) with recurrent neural network (LSTM). In order to check the possible use of these forecasts in different asset management methods, these forecasts are afterwards included into trading signals of investment strategies. As a benchmark method, the Random Walk model producing naive forecasts has been utilized. This research examines daily data from the S&P 500 index for 20 years, from 2000 to 2020, and it includes information on some significant market turbulence. The methods were tested in terms of robustness to changes in parameters and hyperparameters and evaluated based on various error metrics (MAE, MAPE, RMSE MSE). The results show that ARIMA outperforms LSTM in terms of one step ahead forecasts. Finally, LSTM model with a variety of hyperparameters - including a number of epochs, a loss function, an optimizer, activation functions, a number of units, a batch size, and a learning rate - was tested in order to check its robustness.

Keywords: deep learning, recurrent neural networks, ARIMA, algorithmic investment strategies, trading systems, LSTM, walk-forward process, optimization

JEL codes: C4, C14, C45, C53, C58, G13

Note: This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors

Introduction

Predicting the behavior of the stock market includes using time series forecasting as a key component. The unprecedented economic trends and information asymmetry are the main reasons why it is a difficult task. However, the recent growth in algorithmic trading technologies made it quicker and easier to analyze large datasets and provide a forecast with high accuracy. Taking into consideration that investors are highly interested in risk management, they are more likely to invest in a strategy that yields higher risk adjusted returns, and such strategy can be obtained with methods characterized by lower forecasting errors. Hence the accuracy of the forecast plays an important role for the investors.

The main aim of this paper is to investigate which forecasting methods provide the best predictions with regards to lower forecasting errors. A classical model such as ARIMA is compared with a machine learning model such as LSTM which is a type of recurrent neural network. The first research hypothesis of this paper is that *LSTM outperforms ARIMA in terms of one step ahead forecasts* (RH1). The second hypothesis tested in this paper is that *LSTM is robust to changes in the hyperparameters* (RH2), where the hyperparameters were set heuristically at the beginning. Other hypotheses with regards to tuning hyperparameters of the LSTM model are as follows. The third one states that *Increasing the number of epochs leads to better performance of the model, however using more than an optimal number of epochs leads to an over-fitted model* (RH3). The fourth hypothesis is *that using the wrong optimizer and activation function can significantly worsen the accuracy of the LSTM model* (RH4). Additionally, we have added one research of more practical finance nature question (RQ1): *Can we use one step ahead forecasts from ARIMA or LSTM model in buy/sell signals of investment strategies?*

The initial assumption, based on many previous research papers, is that neural networks might, in fact, perform at least as good as the classical method. In order to test this, the research focuses on the performance of each strategy using various error metrics. The study assumes that there is asymmetric information in the stock market which makes it possible to analyze the volatility in the stock prices, as well as predicting the prices within available historical data. This research uses S&P 500 index close prices from 03.01.2000 till 29.06.2020.

The results for Random Walk, ARIMA, and LSTM were obtained using Python in version 3.8.5. Deep learning libraries used for designing, training, and testing the neural network are Keras (version 2.4.3) and TensorFlow (version 1.8.0). All the calculations, as well as graphs and tables were done using Python with Jupyter Notebook (version 6.0.3) development environment. All the relevant libraries and packages were installed using Anaconda Prompt (conda version 4.8.5) on a 64-bit Windows 10 operating system.

This study is divided into five chapters. The first chapter consists of a literature review discussing the previous studies done in the field of automated transactional systems, including the models implemented in this research. The second chapter briefly describes the data used in this study. The third chapter focuses on methodology used for forecasting and error metrics to evaluate the performance of the models. It touches upon time series analysis and the methods of generating a forecast for the implemented models. The fourth chapter presents the empirical results for every method used in this study and compares the performance of each of the forecasts. The fifth chapter consists of sensitivity analysis. It tests the robustness of results by changing parameters and hyperparameters (e.g., changing the size of the training-testing split) and RNN hyperparameters used in the methodology section (e.g., neurons, batch size, epochs,

activation functions, optimizers, etc.). Lastly, the research concludes, verifies hypotheses, and discusses some recommendations for future research.

1. Literature Review

Statistical and machine learning methods are widely discussed in empirical research. The quality of forecast is of great importance to investors around the world. A correct prediction of the stock prices could lead to a greater profit for the investors. In recent times, there has been great improvement in the technology sectors such as increased computational power, which results in a faster calculation in algorithmic trading. The investors are always keen to invest in a strategy that would lead to greater risk-adjusted profit and algorithmic trading can help in such decision makings. Kirilenko and Lo (2013) summarize the fundamental improvement in the field of Finance and the technological improvement that has led to the growth of this field. It occurred mainly due to technological improvements, lower transaction costs, a large volume of trades, and its faster executions, which finally has led to wide-ranging domination of algorithmic trading in 2009 on the NASDAQ, and then it led to the global discussion on the essential changes in regulation. The change was not directed towards enhancements in trading. The changes have essentially been caused by an increment in speed of trading and the volume created by high-frequency trading in all asset markets including equities, currencies, and others (Brogaard et al., 2014).

Several studies show that the economic time series behaves like Random Walk models or at least seem to have Random Walk components. Accepting that certain economic variables behave in a similar way as a Random Walk model is needed as regressing a certain variable against another variable might lead to spurious results. It is mainly due to the fact that the relationship between economic variables is usually observed when, such a relationship does not exist. Similarly, the effects of a temporary shock will not disappear after several years but instead will be permanent (Pindyck and Rubinfeld, 1998). Reilly (1980) and Reynolds et al. (1995) developed methods for automatically identifying ARIMA models for time series (hereafter TS). The method developed by Reynolds et al. (1995) takes into consideration a neural network approach and it is restricted to non-seasonal TS, whereas the method developed by Reilly (1980) works properly for non-seasonal TS but it is a bit less effective when seasonal TS data is concerned. Analytical neural network methods have been widely used in prediction (Chiu et al., 1995; Gao et al., 1997; Cook and Chiu, 1997; Saad et al., 1998).

The recent papers concerning the evaluation of the performance of neural network in pricing liabilities focus mainly on testing various types and architectures of the neural networks, such as the recurrent neural networks (RNNs) and the convolutional neural networks (CNNs). Yang et. al (2017) suggested the use of the gated neural networks, which not only provide reliable prices of the options, but also contain an assurance of economically reasonable and rational results. The model outperforms other network-based models and some of the econometric methods described in the paper. Siami-Namini et al., (2018) has conducted a study on whether and how deep learning-based algorithms for forecasting time series data, such as LSTM, are superior to the traditional algorithms. The ARIMA model has shown great precision and accuracy in terms of predicting the next lags of time series. The empirical studies conducted in this article show that LSTM beats the traditional-based algorithms such as the ARIMA model with an average reduction in error rates obtained by LSTM between 84-87 percent, indicating the superiority of LSTM to ARIMA. Moreover, it was observed that the number of training times ("epoch") has no effect on the performance of the forecast and it shows truly random behavior.

Zenkova and Ślepaczuk (2018) conducted studies on the profitability of an algorithmic trading strategy based on training SVM model for identification of cryptocurrencies with low or high predicted returns. Each cryptocurrency is represented using a set of six technical features: Momentum, Volume change, Relative Strength Index, Force Index, Williams Percent Range and Parabolic stop and reversal system. The performance of the SVM portfolio is compared with the performance of the four benchmark strategies: equally weighted portfolio (EqW), market-cap weighted portfolio (McW) and two buy and hold strategy (one for bitcoin and the second for S&P500 index. It is noted that EqW portfolio outperforms the SVM strategy and all the benchmark strategies. The SVM strategy has not demonstrated any abnormal returns in comparison to benchmark strategies and the algorithm itself does not provide robust outcomes. Bukhari et al. (2020) conducted studies on a novel hybrid model with the strength of fractional order derivative along with their dynamical features of deep learning, long-short term memory (LSTM) networks, in order to predict the abrupt stochastic variation of the financial market. A novel ARFIMA-LSTM hybrid recurrent network together with ARFIMA model-based filters exhibits the linear tendencies better than ARIMA model in the data and passes the residual to the LSTM model which captures nonlinearity in the residual values from exogenous dependent variables. The model overcomes the problem of overfitting and also minimizes the volatility problem. The forecasting performance indicates the effectiveness of the estimated AFRIMA-LSTM hybrid model to improve around 80% of accuracy on RMSE in comparison to traditional forecasting methods.

Qian and Chen (2019) has researched stationary analysis of the stock's time-series data and used the LSTM neural network algorithm in order to predict stock data under various stationary conditions, and conducted statistical analysis on multiple experimental data. Moreover, an ARIMA algorithm was introduced in order to compare with the LSTM. A large number of experimental results show that neural network prediction algorithm - LSTM has a higher prediction accuracy and is not sensitive to the stability response. (Sakshi and A, 2020) have conducted studies on hybrid models using LSTM and ARIMA to capture S&P 500 index using pre-existing Application Programming Interface (API). The Prophet forecasting library created by Facebook which requires less prepossessing has also been used for the purpose of comparison. Prophet has a comparatively high Root Mean Square Error (RMSE) of 27.59 and the Mean Square Error (MSE) of 761.33 whereas the proposed ARIMA- LSTM hybrid has an MSE of 3.03 and RMSE of 1.74 along with a 99% fit of the model, which proves that hybrid performs much better than Prophet forecasting.

Chlebus et al. (2020) tested various machine learning techniques for time series forecasting with the following methods: SVR, KNN, XGBoost, LightGBM, LSTM, ARIMA, ARIMAX with features coming from classes like technical analysis, fundamental analysis, Google Trends entries, markets regarding Nvidia. SVR had the best performance based on stationary attributes. It was also noted that the models based on the stationary variables have a better performance than the models based on stationary and non-stationary variables. Kijewski and Ślepaczuk (2020) compares investment algorithm performance on S&P 500 index with 20 years of data from 2000 to 2020. Several algorithmic strategies including classical methods such as moving average crossover, volatility breakouts, and macroeconomic factors were tested. Statistical and machine learning approaches like ARIMA and LSTM models were also tested respectively. The results show that classical methods using the rolling training-testing window were significantly more robust to changes in hyperparameters than LSTM. Adil and Hamiche (2020) used LSTM to forecast future values for GOOGL and NKE assets using various epochs. It is noted that the number of epochs and the length of the dataset both have a significant impact on testing results. The dataset was tested with various epochs (12 epochs, 25 epochs, 50 epochs and 100 epochs) and it was noticed that training less data with more epochs can significantly

improve the testing results and allows to have better forecasting and prediction values.

Table 1. Significant studies related to various prediction techniques

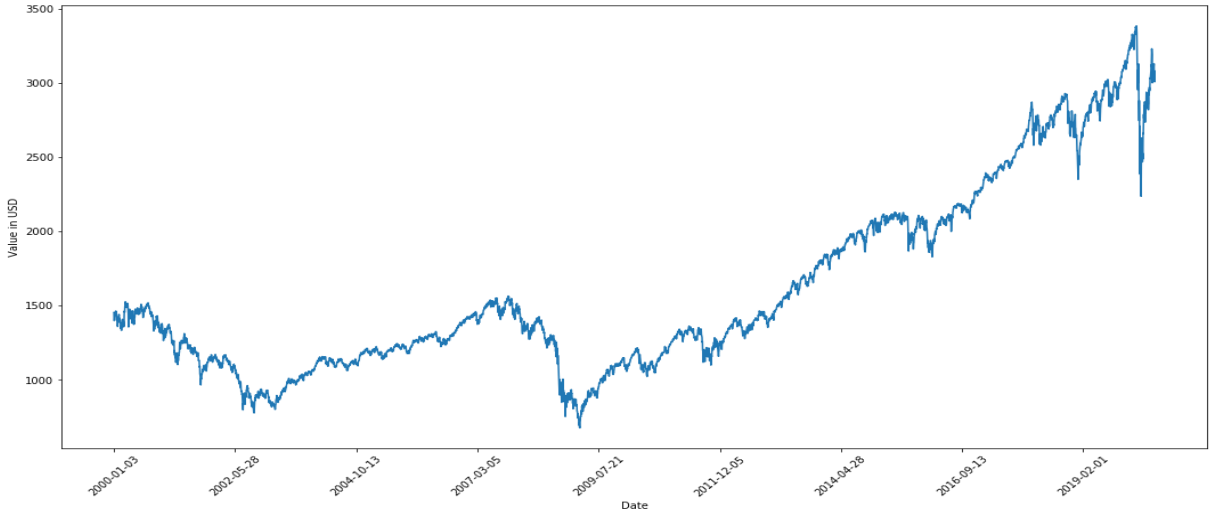
Authors	Dataset	Technique	Evaluation Metrics
Liu et al, 2016	S&P 500	Logistic regression, GDA, Naive Bayes, Linear SVM, RBF SVM, Poly SVM	Accuracy: Logistic regression:60.62%, GDA: 60.62%, Naive Bayes: 60.38%, Linear SVM:59.79%, RBF SVM: 62.51%, Poly SVM: 59.43%
Selvin et al., 2017	NSE	RNN, LSTM, CNN	MAE: 5.13% (RNN), 5.31% (LSTM), 4.98% (CNN)
Maini et al., 2017	Dow Jones Industrial Average	SVM, Random Forest	Accuracy: 84.6% (SVMlinear), 85.18% (SVMRBF) 86.2% (Random forest)
Persio et al., 2017	Google Assets	RNN, LSTM, GRU	Accuracy- 72%
Creighton et al., 2017	S&P 500 and S&P 400	ARIMA- BPNN hybrid	MAE: 16.68, MSE: 434.121 RMSE: 20.836, Accuracy: 45.1%
Deepak et al., 2017	BSE Sensex	SVM- RBF kernel	Accuracy: 80 to 85%
Zhang et al., 2018	TA- Lib	Random Forest	Accuracy: 67.5% Std deviation:3.7%
Baek & Kim, 2018	S&P500 and KOSPI200	LSTM with SingleNet	MSE: 54.1% (S&P500) & 48% (KOSPI200) MAPE: 35.5% (S&P500) & 23.9% (KOSPI200) MAE: 32.7% (S&P500) & 32.7% (KOSPI200)
Wen et al., 2019	S&P 500	CNN using Motif extraction	Accuracy: 56.14%, Precision: 55.44% Recall: 74.75%
Sadia et al., 2019	Kaggle Dataset	SVM, Random Forest	Accuracy: 78.7% (SVM), 80.8% (Random forest)
Mehtab & Sen, 2019	NIFTY 50	LSTM	MAPE: 10.75
Sakshi & A, 2020	S&P 500	Prophet forecasting LSTM-ARIMA hybrid	Prophet: RMSE: 27.59, MSE: 761.33 LSTM-ARIMA hybrid: RMSE: 3.03, MSE: 1.74

Note: GDA: Gaussian Discriminant Analysis, SVM: support vector machine, RBF: Radial Basis Function, RNN: Recurrent neural network, LSTM: Long short-term memory, CNN: Convolutional neural network, GRU: Gated recurrent units, BPNN: Back Propagation Neural Network. ARIMA: Autoregressive integrated moving average.

2. Data Description

This paper uses S&P 500 index close prices from 03.01.2000 till 29.06.2020. The data is downloaded from Yahoo Finance. The S&P 500 is a free float-adjusted weighted stock market capitalization index in the United States. Taking into consideration the United States is one of the financial centers of the world, the S&P 500 is considered one of the most important indexes in the world.

Figure 1. S&P 500 index



Note: S&P 500 index values in US dollars for the period 03.01.2000-29.06.2020.

3. Methodology

3.1. Random Walk model

One of the simplest but yet considered as a very important model in the time series forecasting is the Random Walk model. It assumes that for each period the variable has a state away from the previous value, and each step is independently and identically distributed ("i.i.d."). The first difference of the variable is a series for which the mean model can be applied. So, if the first difference of a time series is an i.i.d. sequence, then it can be said that the Random Walk model is possibly a good candidate.

A Random Walk model can be with "drift" or there might be "no drift" in accordance with the distribution of step sizes having a non-zero mean. Random Walk model *without* drift can be represented as follows:

$$\hat{Y}_{n+k} = Y_n \tag{1}$$

where:

n=number of periods,

k=number of steps ahead forecast,

$\hat{Y}$*=predicted value of y,*

Y=Value at a specific period

The prediction of a Random Walk model without drift says that all future values are equal to the last observed value. However, it does not really mean that all of them are expected to be the same, but they can actually be equally likely to be higher or lower. If the Random Walk model is extrapolated for forecasting the distant future, they are very likely to go off on a horizontal line, which is very similar to the mean model. So, it can be said that they behave like the mean model during long term forecasting.

Random Walk model *with* drift can be represented as follows:

$$\hat{Y}_{n+k} = Y_n + k\hat{d} \tag{2}$$

where:

$\hat{d}$*=estimated drift,*

k=number of steps ahead forecast

The estimated drift $\hat{d}$ is the average increment from one period to the next. Hence the long-term forecast for a Random Walk with drift model appears to be a trend line, having a slope $\hat{d}$. The drift $\hat{d}$ in Random Walk model can be represented as follows:

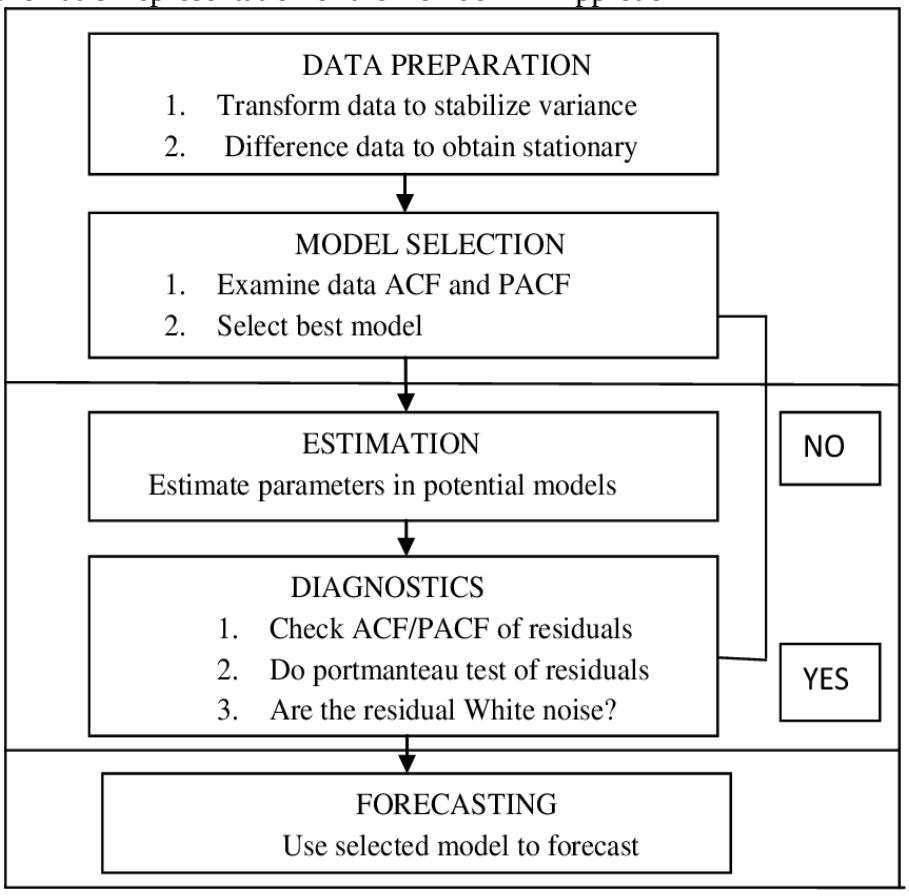
$$\hat{d} = \frac{Y_n - Y_1}{n - 1} \tag{3}$$

It is the slope line in between the first and last data point which is different from the slope line fitted in simple regression. The slope line is then divided by n-1 period to get the estimated drift.

3.2. ARIMA model

Autoregressive Integrated Moving Average (ARIMA) became well known after its wide description in the first edition of Box and Jenkins (1970), where they prepared procedures for specification, estimation diagnostics and forecasting for the univariate time series.

Figure 2. Schematic Representation of the Box-Jenkin Approach



Source: Makridakis et al., 1983

ARIMA (p, d, q) model is divided into three parts:

- autoregressive - adds dependent variables as p-lagged observations,
- integrated - describes d - number of differentiations required to make the time series stationary,
- moving average - adds independent variables as q-lagged error terms.

The model can be described with the following formula:

$$y_t = a_1 y_{t-1} + \cdots + a_p y_{t-p} + e_t + b_1 e_{t-1} + \cdots + b_q e_{t-q} \tag{4}$$

where:

p - is the number of autoregressive lags,

q - is the number of the error lags,

y_t *– is the value of time series at time t,*

e_t *– is the error term at time t.*

There are many methods to calculate the goodness of fit such as *Chi-Squared, Kolmogorov-Smirnoff and Anderson-Darling*, however, the *information criteria* are considered to be one of the most important ones and used in this paper.

AIC: Akaike information criterion (Akaike, 1974):

$$AIC = \left(\frac{2n}{n-k-1}\right) k - 2ln\,[L_{max}] \tag{5}$$

SIC: Schwarz information criterion, also known as Bayesian information criterion BIC (Schwarz, 1997):

$$BIC = ln\,[n]k - 2ln\,[L_{max}\,] \tag{6}$$

HQIC: Hannan-Quinn information criterion (Hannan and Quinn, 1979):

$$HQIC = 2ln\,[ln\,[n]k - 2ln\,[L_{max}] \tag{7}$$

where:

n = number of observations,
k = number of parameters for estimation,
L_{max} *= the maximized value for the log-Likelihood for the estimated model*

The main way to find the optimal ARIMA model is to find the lowest value of the selected information criteria. The $-2ln\,[L_{max}]$term found in each formula is an estimation of the deviation of the model fit. The coefficients k found in the first part in each formula is the degree to which the number of the model parameters that are being penalized.

Once the lowest information criteria are selected, the second step is to estimate the model with the selected parameters using OLS estimation or the maximum likelihood estimation. In this paper, maximum likelihood estimation is selected due to time efficiency. The last step for forecasting in this study is to calculate one step ahead forecast. The detailed procedure is as follows:

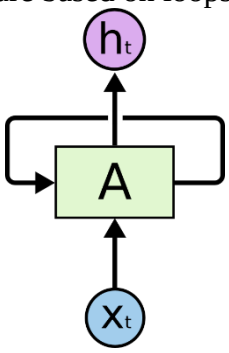
1. Set a training sample for first 4150 days[1] in the dataset
2. Identify the optimal values of p, d, q with p: 0-10, d: 0-3, q: 0-10[2]
3. Estimate the model for selected p, d, q parameters using the maximum likelihood method[3]
4. Forecast one-step ahead
5. Repeat 1-4 points moving the training sample by one day ahead, until the end of the dataset.
6. Calculate the error metrics for the testing dataset (last 1005 days)

3.3. LSTM

3.3.1. Recurrent Neural Network

Recurrent Neural Network unlike the classical Neural network is based on loops as it does not start every time from scratch. It is a network with loops which allows the information to persist.

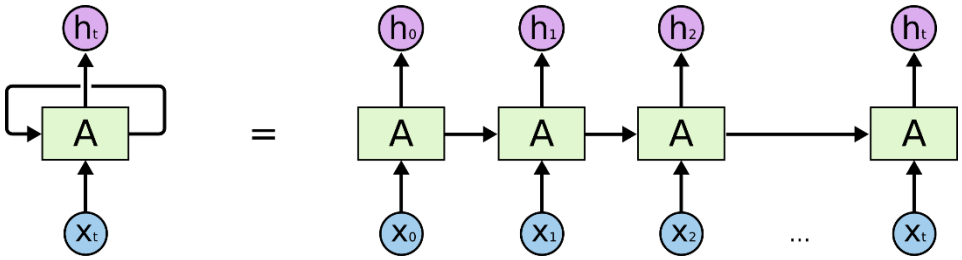
Figure 3. Recurrent Neural Networks are based on loops



Note: Example of single layer in RNN. In the above diagram, x_t are inputs and h_t are output. A is the neural layer. Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ Retrieved on 26.11.2020.

The above picture is a representation of a chunk of neural network. A loop allows this information to pass from one step to the network towards the next step. A recurrent network can be assumed to be multiple copies of the same network and the previous step of the loop helps to predict the next step. Here is what an unrolled recurrent network looks like when the loops are unrolled.

Figure 4. An unrolled recurrent neural network



Note: An unrolled recurrent neural network, where x_t are inputs and h_t are output. A is the neural layer. Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ Retrieved on 26.11.2020.

[1] The length of the in-sample period was determined by the need to cover at least two significant downward (2000-2003, 2007-2009) and upward (2003-2007, 2009-2015) trends of the prices, characterized by different volatility and duration.

[2] The rationale for selecting the range for p and d from 0 to 10 was that we wanted to verify diverse possible values of orders for MA(p) and AR(q) processes. We pursued the same aim when examining the possible values of the order of d in the range d 0 to 3.

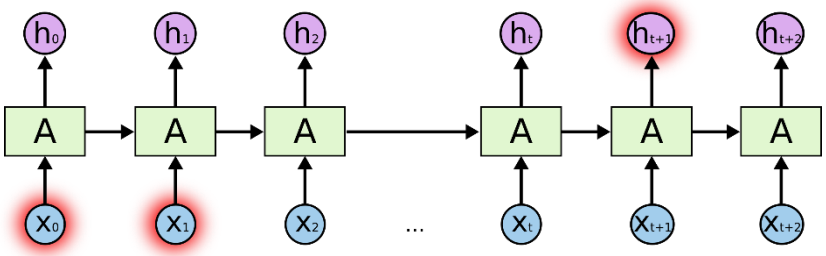
[3] WE did not decide to eliminate the intermediate lags due to the necessity to run hundreds of regressions in an automatic way.

The chain-like Figure 4 reveals that recurrent neural networks are closely related to the sequences and lists. In the last few years, there has been great success applying RNNs towards a variety of problems such as speech recognition, translation, image captioning, stock price forecasting and many others.

3.3.2. Long-Term Dependencies

One of the main problems of RNN is the long-term dependencies. RNN in general connects previous information to predict the next. However, sometimes the most recent information provides the best prediction for the next. However, if there is a considerably large amount of data and the prediction performs between only with the most recent information, the gap grows and RNN finds it harder to connect the information.

Figure 5: Problem with long term dependencies



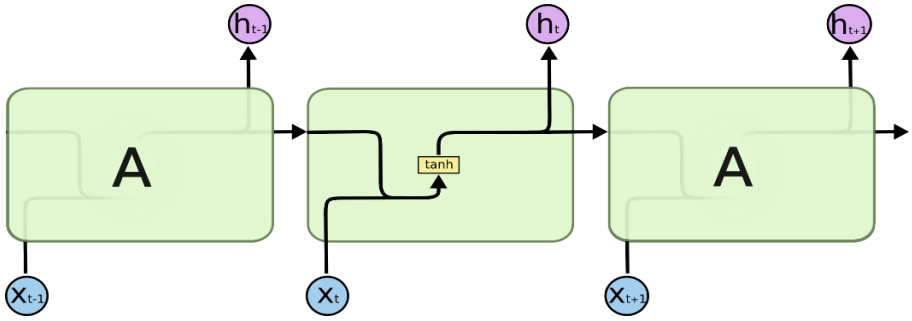
Note: In the above diagram, x_t are inputs and h_t are output. A is the neural layer.
Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ Retrieved on 26.11.2020.

Figure 5 presents a simple representation of the problem with long term dependencies. If x_0 and x_1 are the recurrent inputs used for prediction of h_{t+1}, it might not get the result as per expectation due to the fact that all previous information might not be relevant for performing the present task. Otherwise only the most recent information might be useful. The problem was further explored by (Bengio et al., 1994). However, thanks to LSTM which solves this problem.

3.3.3. LSTM

There are many types of Recurrent Neural network (RNN). Long Short-Term Memory (LSTM) is one of the very popular ones. It is capable of taking data from the past and using this data for future predictions (Patterson and Gibson, 2017). LSTM solves the issues related to long term dependencies. It was introduced by (Hochreiter and Schmidhuber, 1997) and later refined and popularized by many other researchers. Currently, it is widely used and works very well with a large variety of problems.

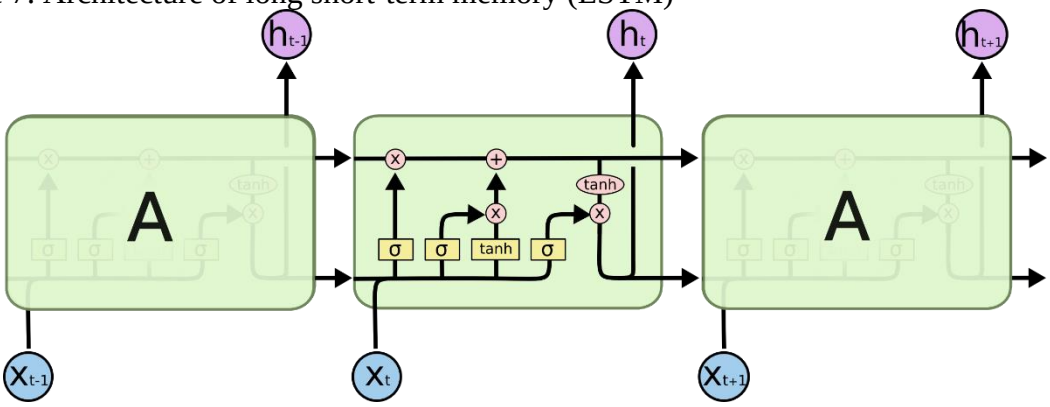
Figure 6. Architecture of the recurrent neural network



Note: Example of a simple recurrent neural network architecture with a single layer and a tanh activation function, where x_t are inputs and h_t are output. A is the neural layer.
Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ Retrieved on 26.11.2020.

RNN is derived by using hidden states from vanilla neural networks, which are a matrix of parameters that are passed between the sequential inputs. The abovementioned is presented in Figure 6 with a horizontal arrow. Taking into consideration that the matrix is passed between the inputs, it allows transferring information from inputs about the importance of the given input, which results in LSTM returning different results for the same input. It takes into consideration the weights of each input.

Figure 7. Architecture of long short-term memory (LSTM)



Note: Example of a multiple layered architecture of LSTM model with input gate, forget gate and output gate, where x_t are inputs and h_t are outputs. A is the neural layer.
Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/, Retrieved on 26.11.2020.

The key to the LSTM model is the cell state shown with the horizontal line running at the top of Figure 7. Even though the LSTM model looks complex at the first glance, the main difference between LSTM and the RNN is that it has three outputs instead of two. The first output is the memory matrix, which is never transformed with non-linear functions, which makes it easier for optimization and solves the problem of gradient vanishing. The second and the third output are the same one and are passed as a classical output from the layer and one is passed towards the next input cells. A single cell in LSTM can be divided into three parts:

1. Forget gate
2. Input gate
3. Output gate

A forget gate is a sigmoid function, which is applied on the concatenation of the input x_t and the previous output h_{t-1}. It returns the matrix of values from range 0 to 1 and then the matrix will be multiplied with the memory matrix. Parameters for which the values are closer to 1 will remain in the memory matrix. The second part Input gate adds new weight with the memory matrix and updates it. Firstly, the sigmoid function decides the impact of each current input data and then the next tanh function applies the non-linear transformation of the modified input data. The weight matrix prepared in such a manner is then added to the memory cell. Lastly, there is an output gate, which outputs the results obtained with the adjusted memory matrix. The main difference between the normal neural network and the LSTM model is that LSTM helps to be able to do weight adjustments made in the memory matrix in accordance with the importance of past information.

3.3.4. Hyperparameters

Deep learning can be a complicated task taking into consideration there are lots of hyperparameters. Taking into consideration the size of the deep learning model such as LSTM, hyperparameter tuning usually takes a considerably long time. There are two ways to do

hyperparameter tuning. The first way is to select one training sample and then use cross-validation in order to solve the best hyperparameter on a training sample and use this single model for the entire testing period. The second is to use heuristic methods and the literature in order to find the optimal hyperparameter. It requires refitting the model several times. The first method is not selected since, in order to get the reliable results using the first method of optimization, the training sample needs to be significantly higher than the testing sample; for instance, to test this method on the last 20 years of data, the model should be tested on the last 80 years of data, which is considerably time-consuming and the stock market might have not had the similar relation as it used to be for such a long period of time. Hence the best method to obtain more accuracy it is better to refit the model by tuning several hyperparameters, which is explored in the sensitivity analysis chapter.

The following hyperparameters were selected using heuristic methods:

- number of units in hidden layers (Neurons): 30
- number of samples in one iteration (Batch size): 64
- activation function used for hidden layer: tanh
- loss function: Mean Squared Error
- optimizer: Adam
- epochs: 100
- learning rate: 0.001
- default dropout rate for LSTM layer: 0

3.4. Performance measures

The predictive ability of each model was evaluated using four error metrics:

1. Mean absolute error (MAE),
2. Mean absolute percentage error (MAPE),
3. Mean squared error (MSE) and
4. Root-mean-square error (RMSE).

The formulas are as follows:

Mean absolute error (MAE): The n-period average of the absolute difference between the actual value and predicted value.

$$MAE = \frac{1}{n}\sum_{t=1}^{n} |y_t - \hat{y}_t| \tag{8}$$

where:

$\hat{y}_t$ - *predicted value of y*

y_t - *actual value*

Mean absolute percentage error (MAPE): Mean absolute error in percentage terms

$$MAPE = \frac{1}{n}\sum_{t=1}^{n} \left|\frac{y_t - \hat{y}_t}{y_t}\right| \times 100\% \tag{9}$$

Mean squared error (MSE): The n-period average squared difference between the actual value and the predicted value.

$$MSE = \frac{1}{n}\sum_{t=1}^{n}(y_t - \hat{y}_t)^2 \tag{10}$$

Root-mean-square error (RMSE): Square root of MSE.

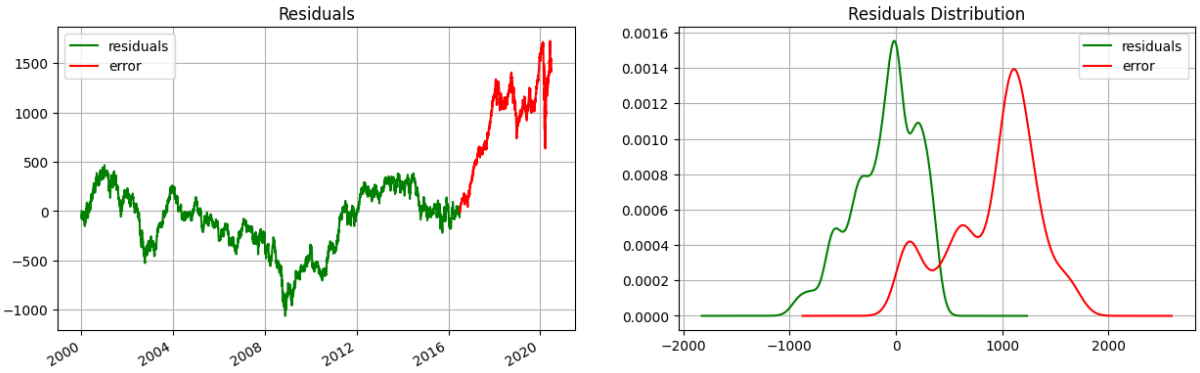
$$RMSE = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(y_t - \hat{y}_t)^2} \tag{11}$$

4. Empirical Results

4.1. Random Walk Model

Random Walk model generating naïve forecasts has been used as a benchmark strategy in order to compare other strategies. The residual refers to the difference between the observed value and the actual value of the model. The error refers the difference between the forecasted value and the actual value of S&P500 time series data, which has been used for the purpose of comparison with other models. Looking at Figure 8 and Table 2, we can observe very poor performance.

Figure 8. Random Walk Model Simulation



Note: Simulation of Random Walk Model where *ts* refers to time series of S&P500 index.

Table 2. Error metrics for Random Walk model

Model Name	MAE	MAPE	MSE	RMSE
Random Walk	958.36	34.42	1135820.64	1066.57

Note: Random Walk model results for S&P 500 index.

4.2. ARIMA

This strategy has been optimized using the lowest AIC, BIC and HQIC criteria. Table 2 shows that the lowest AIC observed is 34364.292, BIC is 34379.615 and HQIC is 34363.252. The best p, d, q values for ARIMA is ARIMA (1,1,1) which has been used during this study.

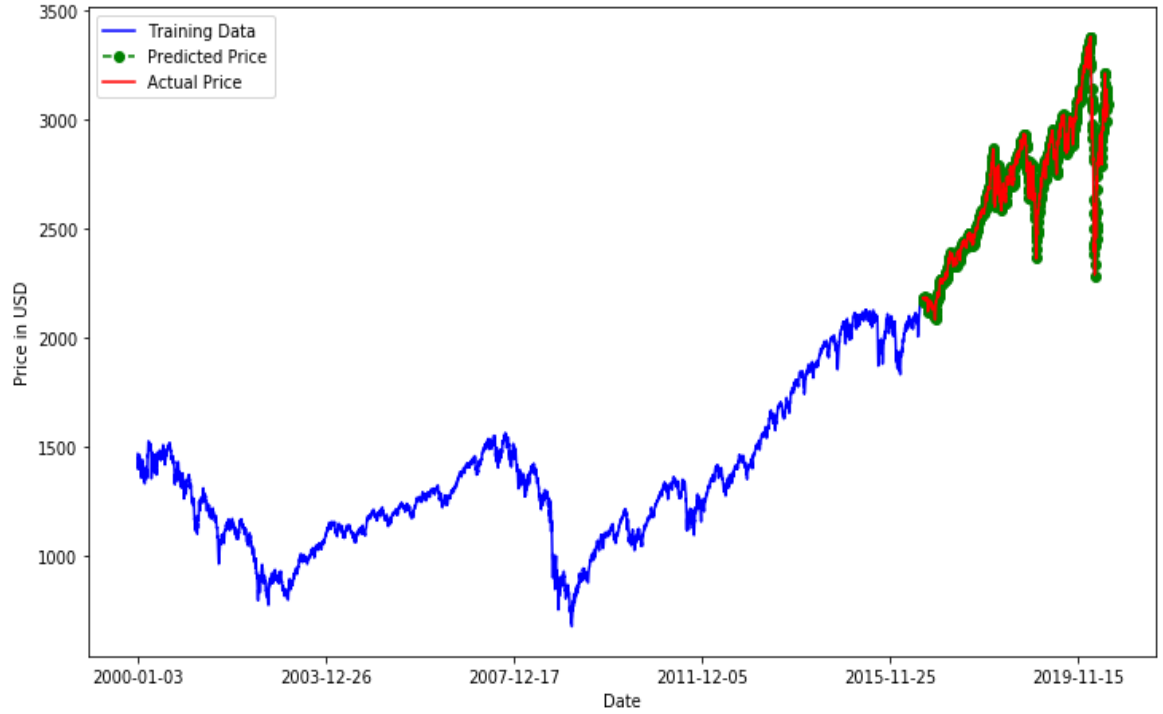
Table 3. Statistical measures of fit - Information Criteria

ARIMA (1,1,1)		
AIC	BIC	HQIC
34354.292	34379.615	34363.252

Note: The lowest measures of information criteria for ARIMA (1,1,1) model.

The best p, d, q values for ARIMA trained on S&P 500 dataset from 03.01.2000-30.06.2016 have been implemented for testing dataset from 01.07.2016 till 29.06.2020 (total 1005 days). Figure 8 shows that ARIMA (1,1,1) forecasting performed very well in comparison to the Random Walk model for one step ahead forecasting. Training and testing size have been changed in the sensitivity analysis section to check how it performed during the 2008 financial crisis as well as during COVID time in 2020. Main forecast error metrics are summarized in Table 4, where we see much lower values for ARIMA model when compared with Random Walk model.

Figure 9. The forecasts of ARIMA (1,1,1) model for S&P 500 index.



Note: Training period: the first 4150 days and testing period: the last 1005 days of S&P 500 index.

Table 4. Error metrics for ARIMA (1,1,1) model.

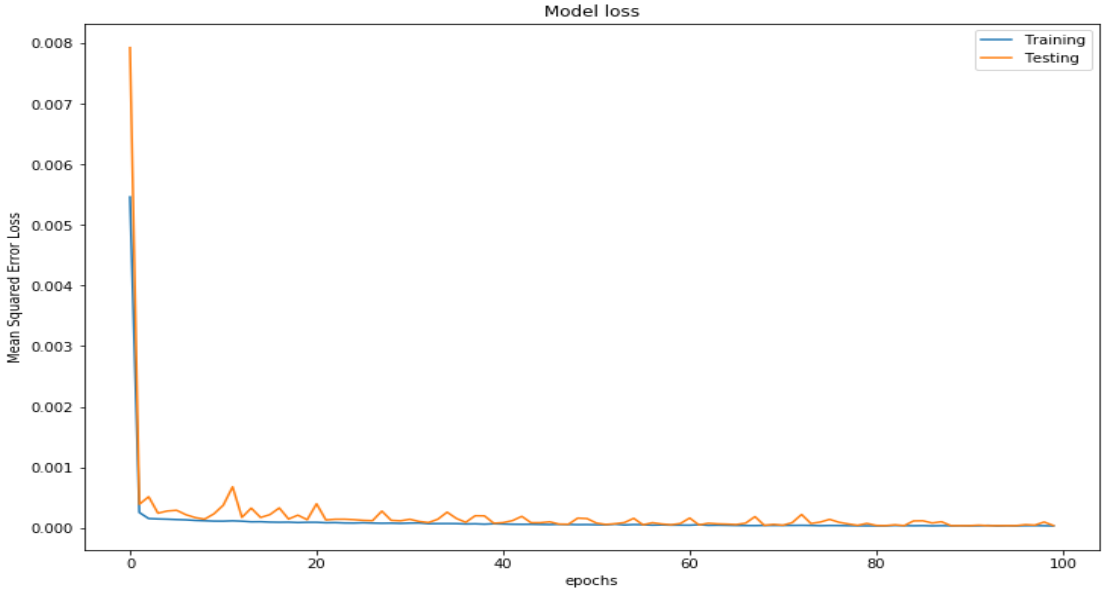
Model Name	MAE	MAPE	MSE	RMSE
Random Walk	958.36	34.42	1135820.64	1066.57
ARIMA	**16.96**	**0.13**	**741.72**	**27.23**

Note: Training was performed on the first 4150 days and testing on the last 1005 days of S&P 500 Index. Bolded numbers indicate the lowest value of error metric.

4.3. LSTM

Figure 10 shows the MSE loss function for the LSTM model with regards to the number of epochs. As the number of the epochs increases, it is noted that the losses for both the training and testing set decreases and when it approaches near 100 epochs, the loss fluctuations observed is closer to 0. More than 100 epochs might lead to overfitting of the model, which is testing during the sensitivity analysis section.

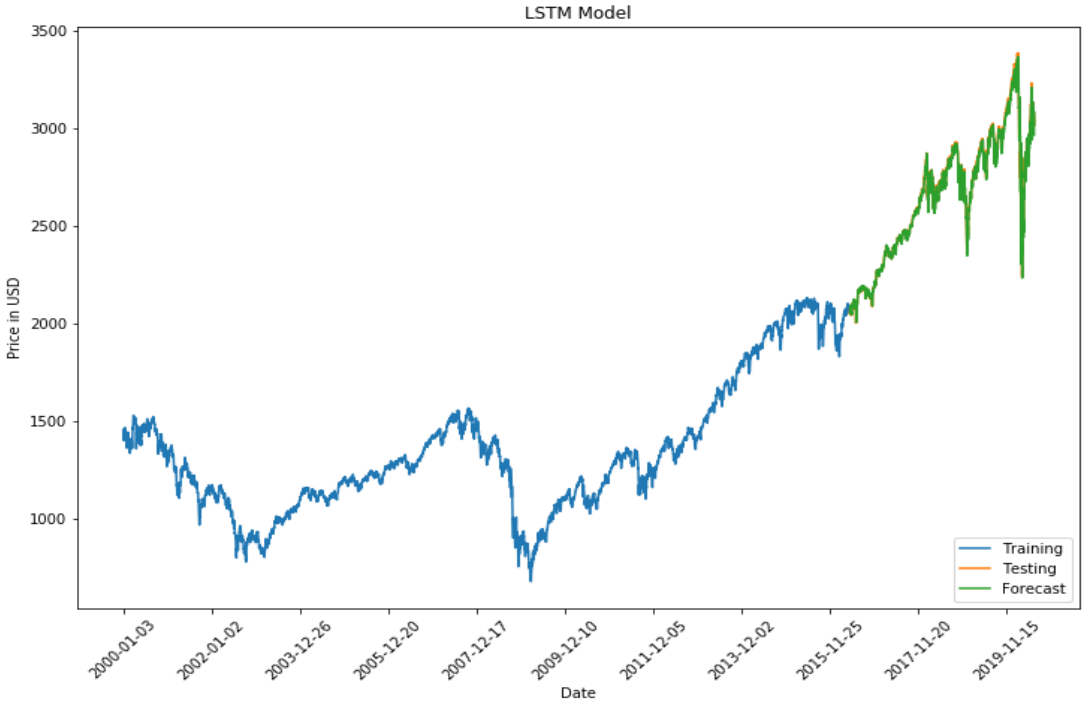
Figure 10. MSE for LSTM model with various number of epochs.



Note: LSTM strategy with the following hyperparameters: Training period: the first 4150 days and testing: the last 1005 days of S&P 500 index. Hyperparameters used: Number of units in hidden layers: 30, Batch size: 64, activation function used for hidden layer: tanh, loss function: Mean Squared Error, optimizer: Adam, epochs: 100, dropout rate: 0, learning rate: 0.001.

Figure 11 presents forecasted values for S&P 500 index with LSTM models. It can be noted that LSTM provided considerably good predictions with very low forecast errors, which are additionally presented in Table 5.

Figure 11. The forecasts of LSTM model for S&P 500 index.



Note: The following hyperparameters were used: Training period: the first 4150 days and testing: the last 1005 days of S&P 500 index. Hyperparameters used: Training: First 4150 datasets and testing last 1005 datasets of S&P 500 Index. Hyperparameters used: Number of units in hidden layers: 30, Batch size: 64, activation function used for hidden layer: tanh, loss function: Mean Squared Error, optimizer: Adam, epochs: 100, dropout rate: 0, learning rate: 0.001.

Table 5. Error metrics for LSTM model.

Model Name	MAE	MAPE	MSE	RMSE
Random Walk	958.36	34.42	1135820.64	1066.57
LSTM	**21.77**	**0.81**	**1399.43**	**37.40**

Note: Error metrics for LSTM model. The following hyperparameters were used: Training period: the first 4150 days and testing: the last 1005 days of S&P 500 index. Hyperparameters used: Number of units in hidden layers: 30, Batch size: 64, activation function used for hidden layer: tanh, loss function: Mean Squared Error, optimizer: Adam, epochs: 100, dropout rate: 0, learning rate: 0.001. Bolded numbers indicate the lowest value of error metric.

4.4. Comparison

Overall, both tested models were quite good at one step ahead forecasting. As for comparison, ARIMA (1,1,1) overall performed better than the LSTM model what is summarized in Table 6.

Table 6. The comparison of error metrics for ARIMA (1,1,1) and LSTM models.

Model Name	MAE	MAPE	MSE	RMSE
Random Walk	958.36	34.42	1135820.64	1066.57
ARIMA	**16.96**	**0.13**	**741.72**	**27.23**
LSTM	21.77	0.81	1399.43	37.40

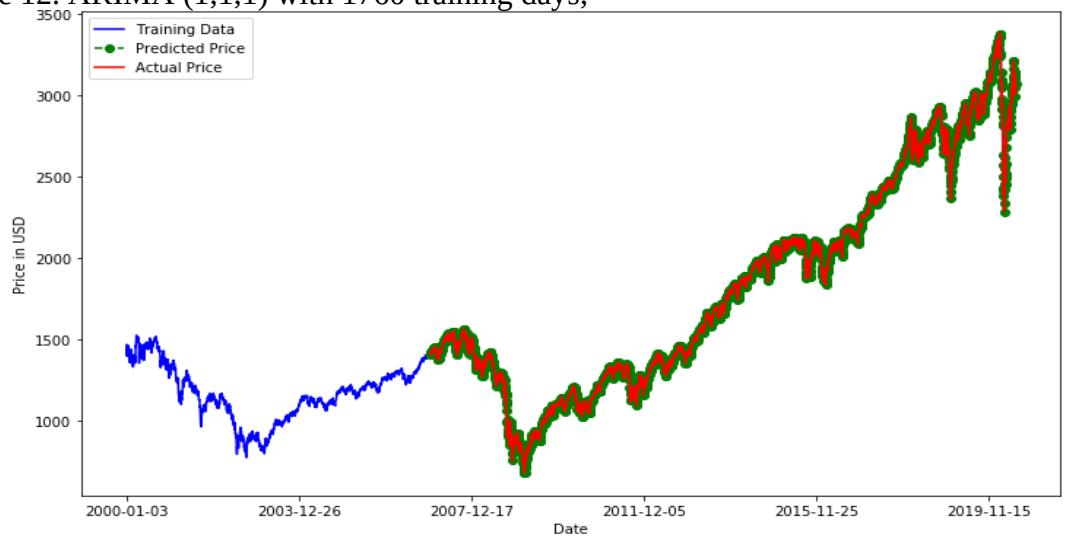
Note: ARIMA (1,1,1) uses the same training and testing datasets: Training: First 4150 days and testing last 1005 days of S&P 500 index. Hyperparameters used: Number of units in hidden layers: 30, Batch size: 64, activation function used for hidden layer: tanh, loss function: Mean Squared Error, optimizer: Adam, epochs: 100, dropout rate: 0, learning rate: 0.001. Bolded numbers indicate the lowest value of error metric.

5. Sensitivity Analysis

5.1. ARIMA Sensitivity

Figure 12 shows ARIMA (1,1,1) with 1760 training days and 3395 testing days. The testing dataset starts from the start of 2007 to check how the model performed during the financial crisis and afterwards. Figure 12 and Table 7 show that it performed considerably well and the financial crisis does not impact much one-step forecasting. Taking into consideration, that testing data includes not only the financial crisis, but also the period after the financial crisis, where the market was in general upward trending, such good performance makes sense.

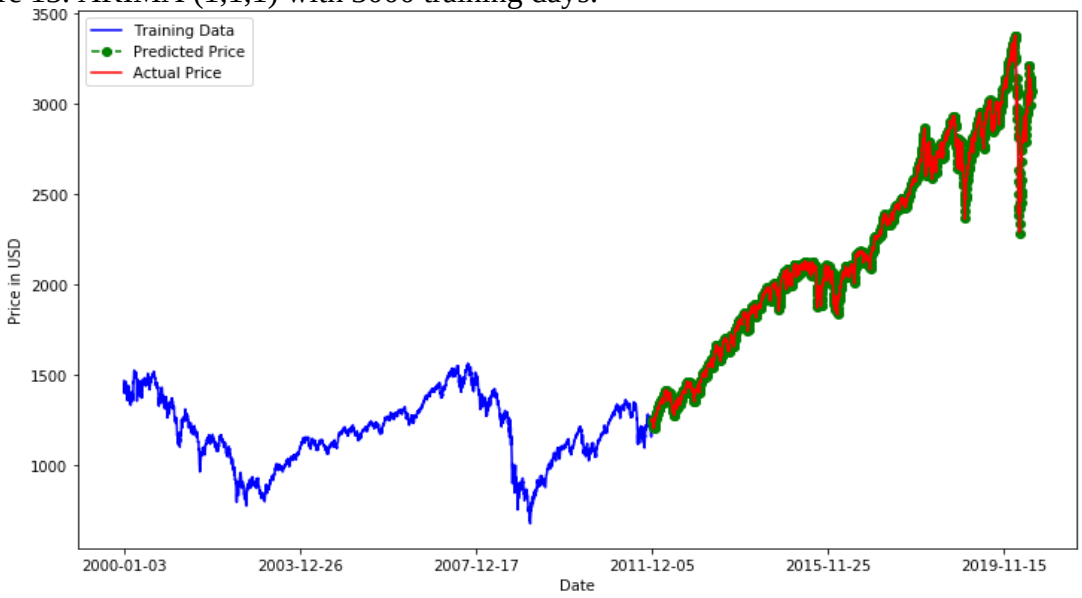
Figure 12. ARIMA (1,1,1) with 1760 training days,



Note: The training dataset was set to the first 1760 observations for the S&P 500 index to see how it performed during and after the 2007-2008 financial crisis and afterwards.

Figure 13 shows ARIMA (1,1,1) with 3000 training days. It is chosen randomly to check how well it is able to predict during non-crisis time. Figure 13 and Table 7 show that it performed considerably well for one-step forecasting.

Figure 13. ARIMA (1,1,1) with 3000 training days.



Note: The training dataset consists of the first 3000 observations for S&P 500 index.

Figure 14 shows ARIMA (1,1,1) with 5093 training datasets. The testing dataset starts from the start of 2020 to check how the model performed during the COVID-19. Figure 14 and Table 7 show that it performed the worst in comparison to other training datasets, which is reasonable. However, the impact was not very significant during one step ahead forecasting.

Figure 14. ARIMA (1,1,1) with 5093 training days.

Note: The training dataset was set to the first 5093 observations for the S&P 500 index to see how it performed during COVID-19.

Table 7. Error metrics for ARIMA (1,1,1) model with various length of training period.

Model Name	MAE	MAPE	MSE	RMSE
Random Walk	958.36	34.42	1135820.64	1066.57
ARIMA	16.96	0.13	741.72	27.23
ARIMA 1760 training days	**12.87**	0.45	**391.98**	**19.80**
ARIMA 3000 training days	13.34	0.30	446.48	21.13
ARIMA 5093 training days	39.98	**0.07**	2700.33	51.96

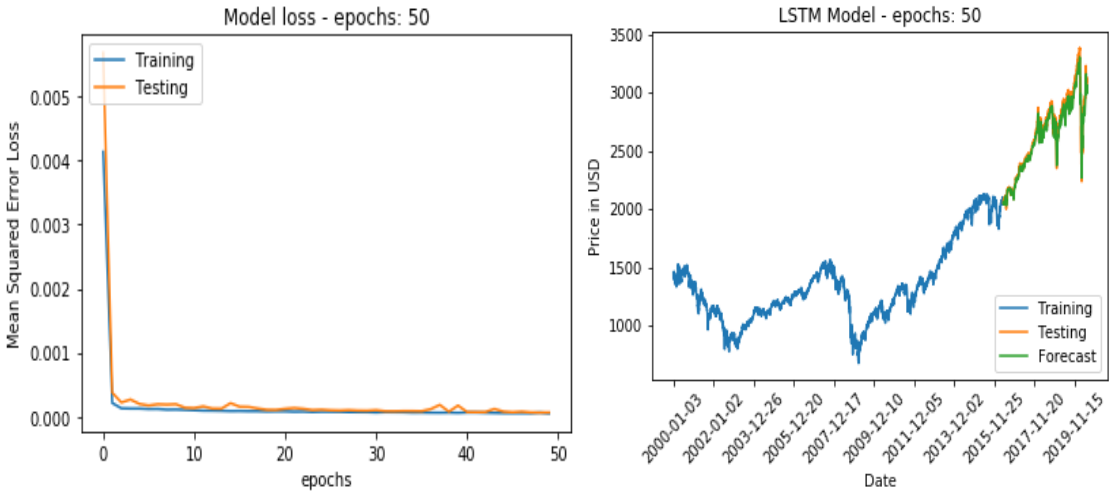
Note: Dataset used: S&P 500 Index from the beginning of 2000 till mid-2020. 1760 training days period is used to check the behaviour of the ARIMA model during and after the 2007–2008 financial crisis. 3000 training days period is chosen randomly and 5093 training days period is used to check the behaviour of the ARIMA model during the 2020 COVID-19. Initial training days period: 4150 days. Bolded numbers indicate the lowest value of error metric.

5.2. LSTM Sensitivity

5.2.1. Epochs

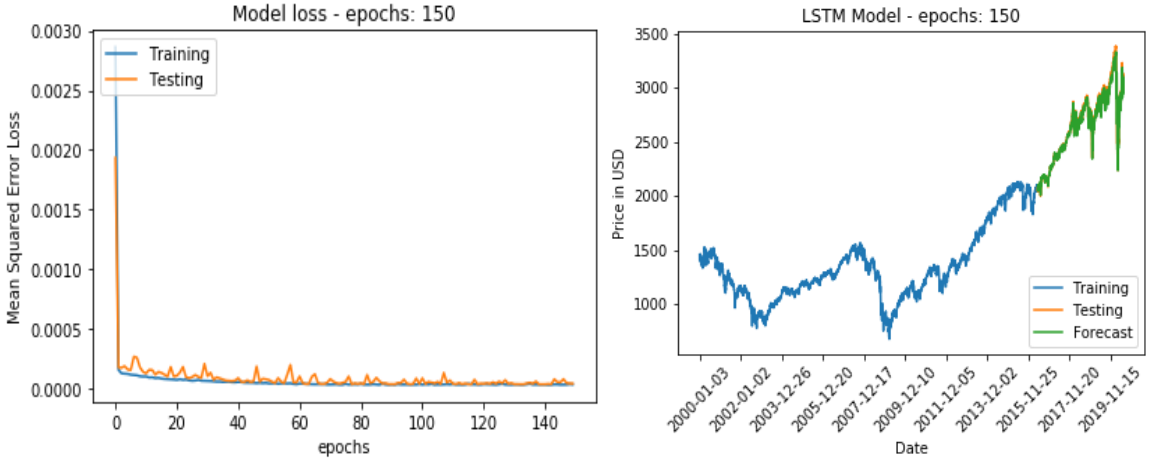
The first tested hyperparameter is the number of epochs. A line plot is created which shows the MSE loss function over the number of epochs for both the training (blue) and testing (orange) sets. MSE is the default loss function used for the LSTM model. Mathematically, it is a preferred loss function under the inference framework of maximum likelihood when the distribution of the target variable is Gaussian. In all the epochs evaluated, it can be seen that the model learned the problem achieving zero error or at least to three decimal places. Looking at Figure 15, 16 and 17 and Table 8, it can be seen that the model performed very well with 100 epochs and worst with 50 epochs. However, increasing the epochs to 300 leads to overfitting of the data.

Figure 15. LSTM model with 50 epochs



Note: Sensitivity analysis of epochs in LSTM strategy. The following hyperparameters were used: Training: First 4150 days and testing last 1005 days of S&P 500 index. Hyperparameters used: Number of units in hidden layers: 30, Batch size: 64, activation function used for hidden layer: tanh, loss function: Mean Squared Error, optimizer: Adam, epochs: 50, dropout rate: 0, learning rate: 0.001.

Figure 16. LSTM model with 150 epochs



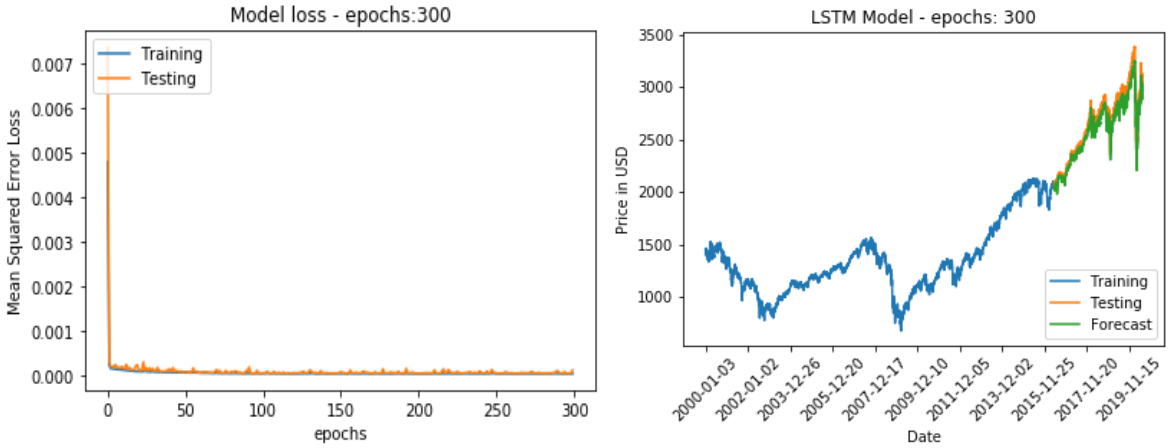
Note: Sensitivity analysis of epochs in LSTM strategy. The following hyperparameters were used: Training: First 4150 days and testing last 1005 days of S&P 500 index. Hyperparameters used: Number of units in hidden layers: 30, Batch size: 64, activation function used for hidden layer: tanh, loss function: Mean Squared Error, optimizer: Adam, epochs: 150, dropout rate: 0, learning rate: 0.001.

Table 8. Error metrics for LSTM model with various number of units.

Model Name	MAE	MAPE	MSE	RMSE
Random Walk	958.36	34.42	1135820.64	1066.57
LSTM	**21.77**	**0.81**	**1399.43**	**37.40**
LSTM: 50 epochs	37.26	1.37	2453.63	49.53
LSTM: 150 epochs	26.87	0.98	1759.62	41.95
LSTM: 300 epochs	65.95	2.48	5862.99	76.57

Note: Error metrics for various number of epochs in LSTM strategy. The following changes in the number of epochs were tested: 50 epochs, 150 epochs, 300 epochs. Initial epochs: 100. Bolded numbers indicate the lowest value of error metric.

Figure 17. LSTM model with 300 epochs



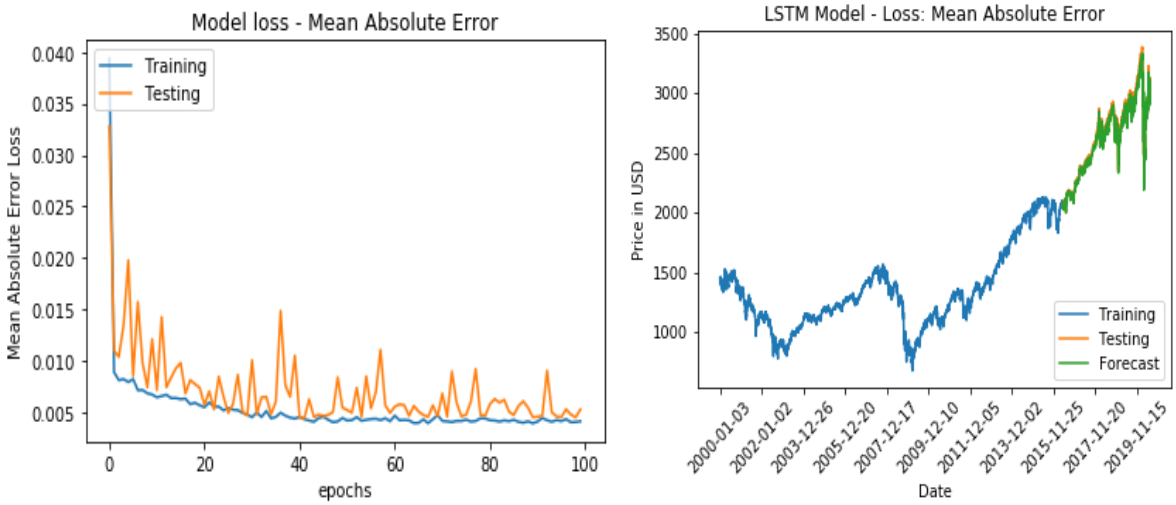
Note: Sensitivity analysis for various number of epochs in LSTM strategy. The following hyperparameters were used: Training: Note: Training: First 4150 days and testing last 1005 days of S&P 500 index. Hyperparameters used: Number of units in hidden layers: 30, Batch size: 64, activation function used for hidden layer: tanh, loss function: Mean Squared Error, optimizer: Adam, epochs: 300, dropout rate: 0, learning rate: 0.001.

5.2.2. Loss Function

The next tested hyperparameter is the loss function. Figures 18, 19, 20 and Table 9 present results using this changed hyperparameter. Three options were tested for loss function: MAE,

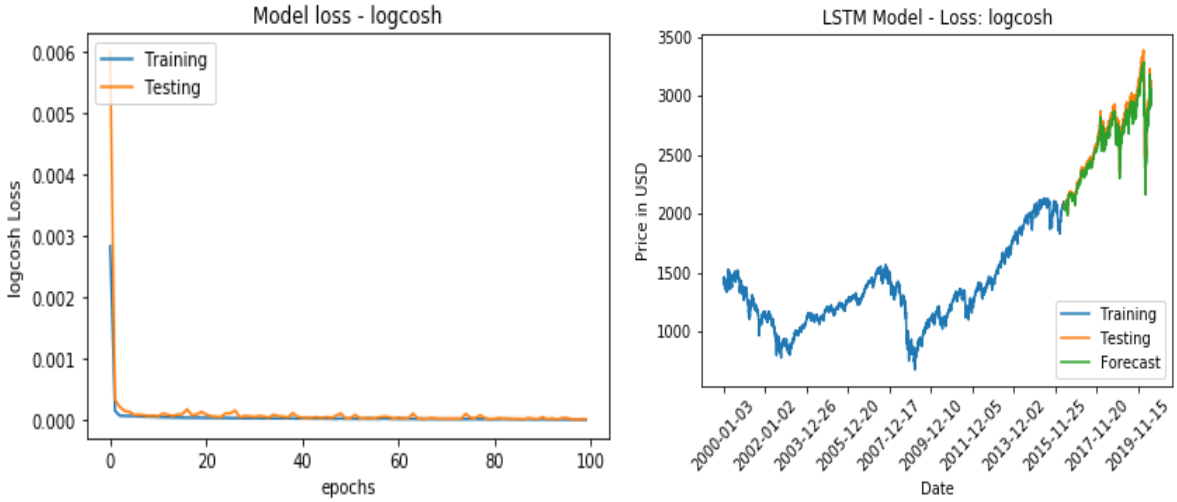
logcosh and huber loss. Initially, the loss function was set to be MSE. LSTM with huber loss performed the best among these three options. However, none of them beat the initial MSE. There is not a single loss function which works for all the data types. It depends on many factors including the presence of outliers in the data, the choice of the machine learning algorithm, the time efficiency of the gradient descent in the data, the ease of finding the derivatives and the confidence of predictions. Having in mind that huber loss takes into account the outliers and the data had 258 outliers, the result obtained makes sense.

Figure 18. LSTM model with loss function: Mean Absolute Error (MAE)



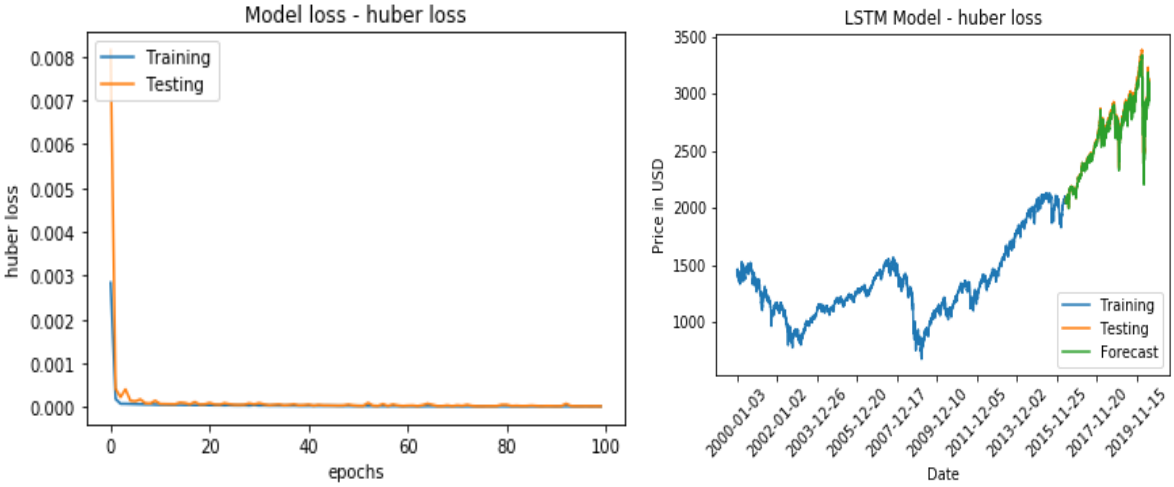
Note: Sensitivity analysis of loss function in LSTM strategy. The following hyperparameters were used: Training period: the first 4150 days and testing: the last 1005 days of S&P 500 index. Hyperparameters used: Number of units in hidden layers: 30, Batch size: 64, activation function used for hidden layer: tanh, loss function: Mean Absolute Error, optimizer: Adam, epochs: 100, dropout rate: 0, learning rate: 0.001.

Figure 19. LSTM model with loss function: logcosh



Note: Sensitivity analysis of loss function in LSTM strategy. The following hyperparameters were used: Training period: the first 4150 days and testing: the last 1005 days of S&P 500 index. Hyperparameters used: Number of units in hidden layers: 30, Batch size: 64, activation function used for hidden layer: tanh, loss function: logcosh, optimizer: Adam, epochs: 100, dropout rate: 0, learning rate: 0.001.

Figure 20. LSTM model with loss function: huber loss



Note: Note: Sensitivity analysis of loss function in LSTM strategy. The following hyperparameters were used: Training period: the first 4150 days and testing: the last 1005 days of S&P 500 index. Hyperparameters used: Number of units in hidden layers: 30, Batch size: 64, activation function used for hidden layer: tanh, loss function: huber loss, optimizer: Adam, epochs: 100, dropout rate: 0, learning rate: 0.001.

Table 9. Error metrics for LSTM model with various loss functions.

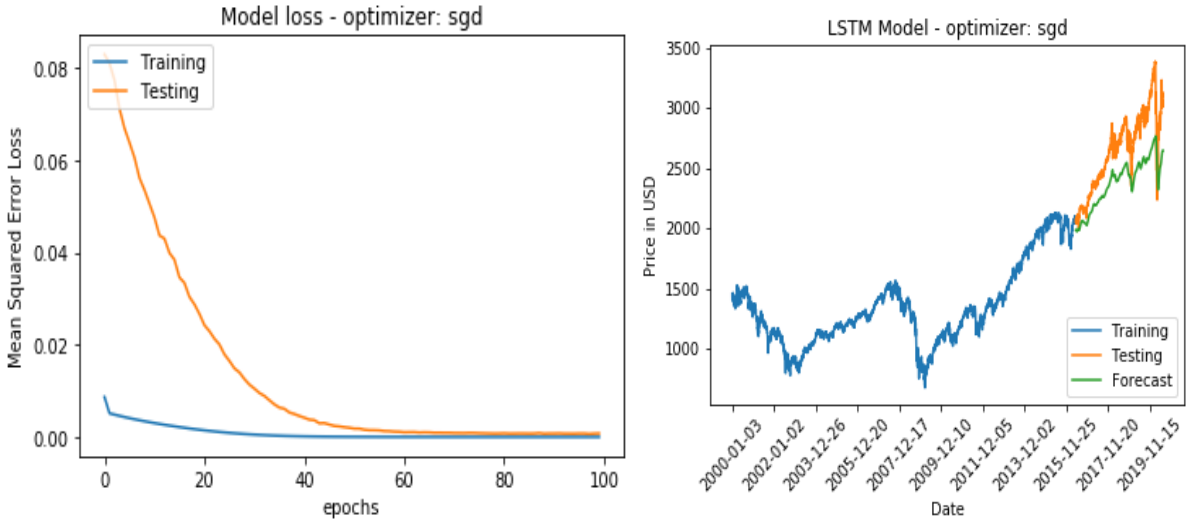
Model Name	MAE	MAPE	MSE	RMSE
Random Walk	958.36	34.42	1135820.64	1066.57
LSTM	**21.77**	**0.81**	**1399.43**	**37.40**
LSTM: Loss function: MAE	37.40	1.39	2736.53	52.31
LSTM: Loss function: logcosh	49.42	1.84	3685.27	60.71
LSTM: Loss function: huber loss	29.88	1.10	2020.87	44.95

Note: Error Metrics for the loss function in LSTM strategy. The following changes in loss functions were tested: MAE loss, logcosh loss and huber loss. Initial loss function: MSE. Bolded numbers indicate the lowest value of error metric.

5.2.3. Optimizer

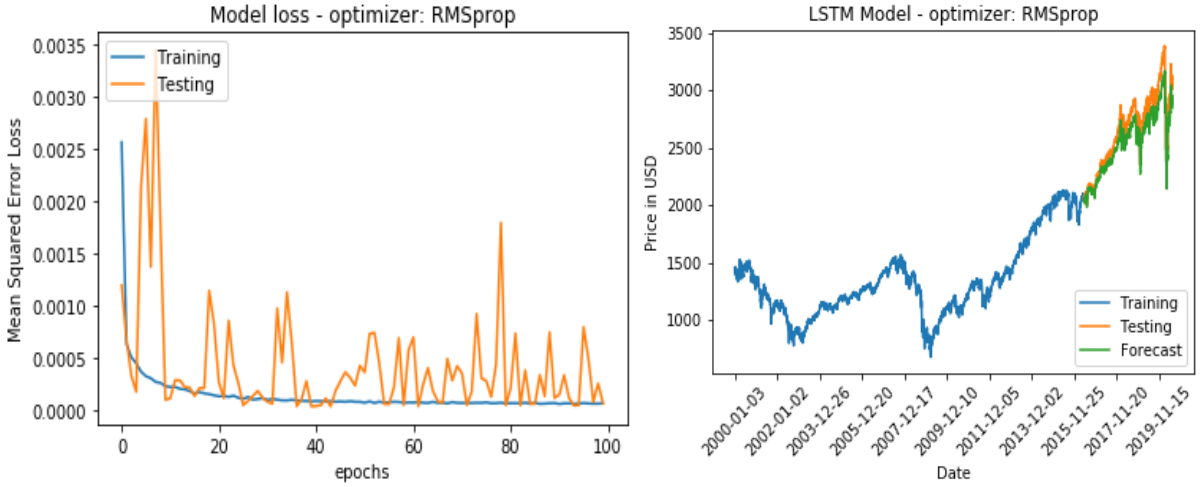
The next tested hyperparameter is the optimizer. Figures 21, 22, 23 and Table 10 present the result using this changed hyperparameter. The following optimizers were tested: sgd, RMSprop and adagrad. Initially the optimizer was set to be adam. The results obtained were not robust to changes in the optimizers. The optimization process is important in terms of achieving accurate results. It is noted that the model performed best with the RMSprop optimizer among three of these tested hyperparameters. It performed worst with sgd and adagrad. However, none of them beat adam optimizer that was set at the beginning.

Figure 21. LSTM model with sgd optimizer



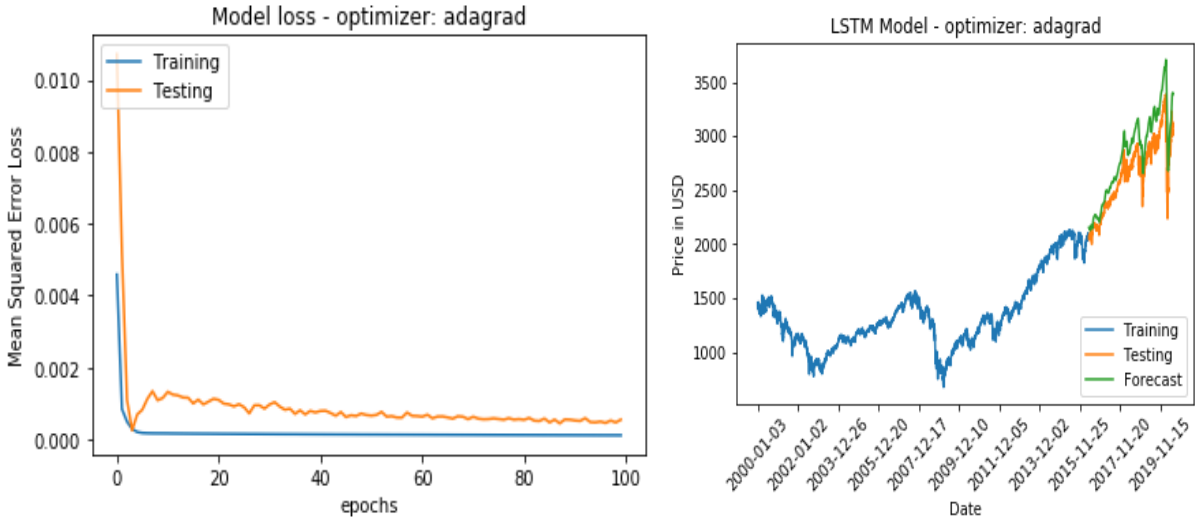
Note: Sensitivity analysis of optimizer in LSTM strategy. The following hyperparameters were used: Training period: the first 4150 days and testing: the last 1005 days of S&P 500 index. Hyperparameters used: Number of units in hidden layers: 30, Batch size: 64, activation function used for hidden layer: tanh, loss function: Mean Squared Error, optimizer: sgd, epochs: 100, dropout rate: 0, learning rate: 0.001.

Figure 22. LSTM model with RMSprop optimizer



Note: Sensitivity analysis of optimizer in LSTM strategy. The following hyperparameters were used: Training period: the first 4150 days and testing: the last 1005 days of S&P 500 index. Hyperparameters used: Number of units in hidden layers: 30, Batch size: 64, activation function used for hidden layer: tanh, loss function: Mean Squared Error, optimizer: RMSprop, epochs: 100, dropout rate: 0, learning rate: 0.001.

Figure 23. LSTM model with adagrad optimizer



Note: Sensitivity analysis of optimizer in LSTM strategy. The following hyperparameters were used: Training period: the first 4150 days and testing: the last 1005 days of S&P 500 index. Hyperparameters used: Number of units in hidden layers: 30, Batch size: 64, activation function used for hidden layer: tanh, loss function: Mean Squared Error, optimizer: adagrad, epochs: 100, dropout rate: 0, learning rate: 0.001.

Table 10. Error metrics for LSTM model with various optimizers.

Model Name	MAE	MAPE	MSE	RMSE
Random Walk	958.36	34.42	1135820.64	1066.57
LSTM	**21.77**	**0.81**	**1399.43**	**37.40**
LSTM: Optimizer: sgd	281.03	11.54	95555.80	309.12
LSTM: Optimizer: RMSprop	105.16	3.98	14158.35	118.99
LSTM: Optimizer: adagrad	184.50	6.27	43970.19	209.69

Note: Error Metrics for various optimizer in LSTM strategy. The following changes in optimizers were tested: sgd, RMSprop, adagrad. Initial optimizer: Adam. Bolded numbers indicate the lowest value of error metric.

5.2.4. Activation Functions

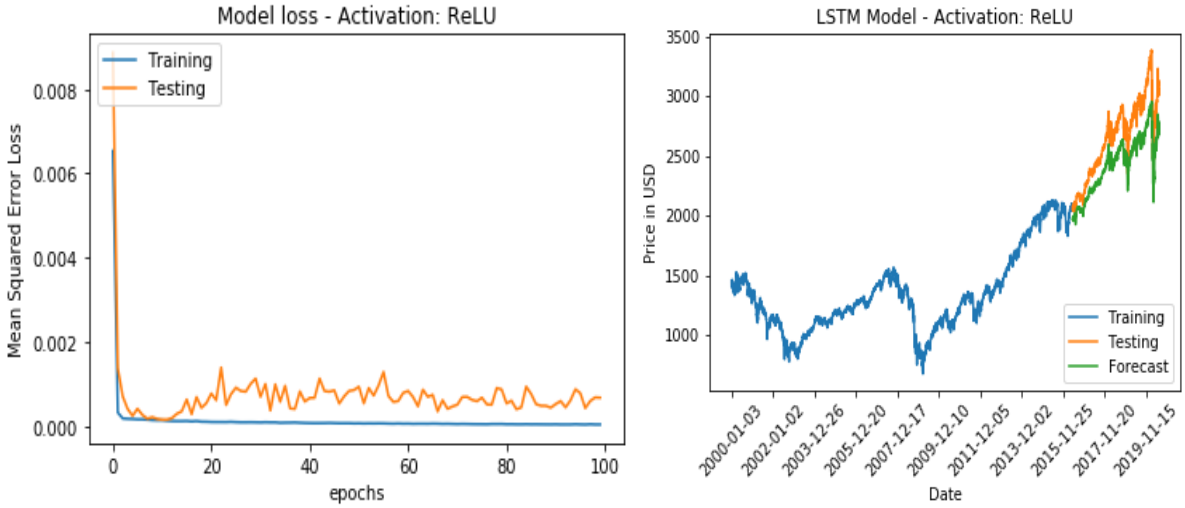
Next tested hyperparameter is the activation function. Figure 24, 25, 26 and Table 11 use these changed hyperparameters. The following activation functions were tested: ReLU, ELU and sigmoid. Initially, the activation function was set to be tanh. The activation function: ELU performed best among these three tested whereas ReLU and sigmoid performed the worst and none of the activation functions beat tanh, which was set at the beginning. However, the results obtained are not robust to changes in the hyperparameters. -

Table 11. Error metrics for LSTM Model with various activation functions.

Model Name	MAE	MAPE	MSE	RMSE
Random Walk	958.36	34.42	1135820.64	1066.57
LSTM	**21.77**	**0.81**	**1399.43**	**37.40**
LSTM: Activation: ReLU	226.45	9.11	58650.27	242.18
LSTM: Activation: ELU	67.27	2.54	5822.19	76.30
LSTM: Activation: sigmoid	230.19	9.21	64842.60	254.64

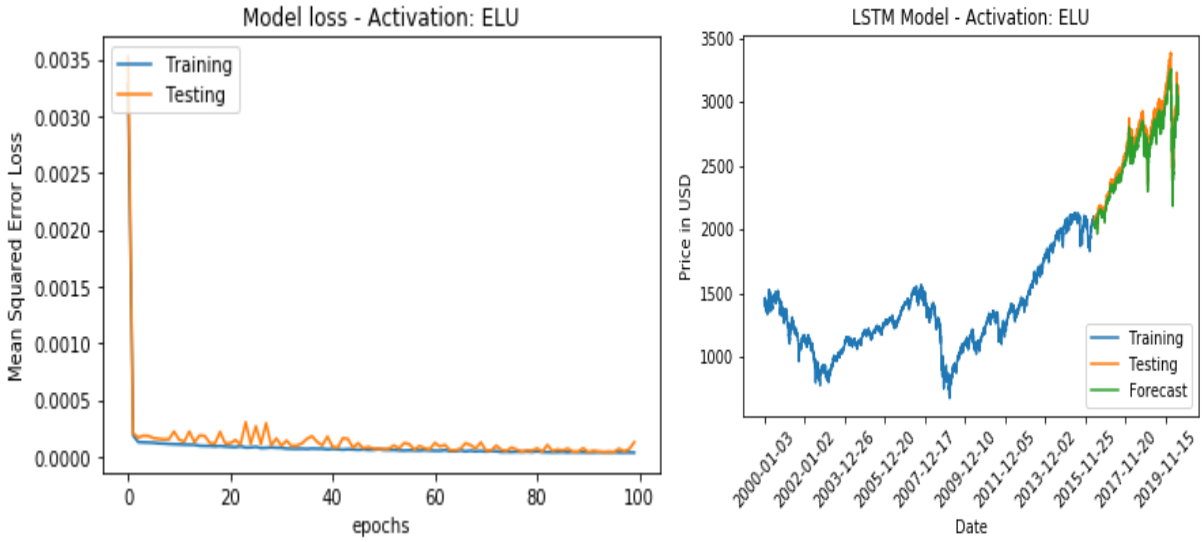
Note: Error metrics sensitivity analysis for various activation functions in LSTM strategy. The following changes in activation functions were tested: ReLU, ELU, sigmoid. Initial activation function: tanh. Bolded numbers indicate the lowest value of error metric.

Figure 24. LSTM model with ReLU activation functions.



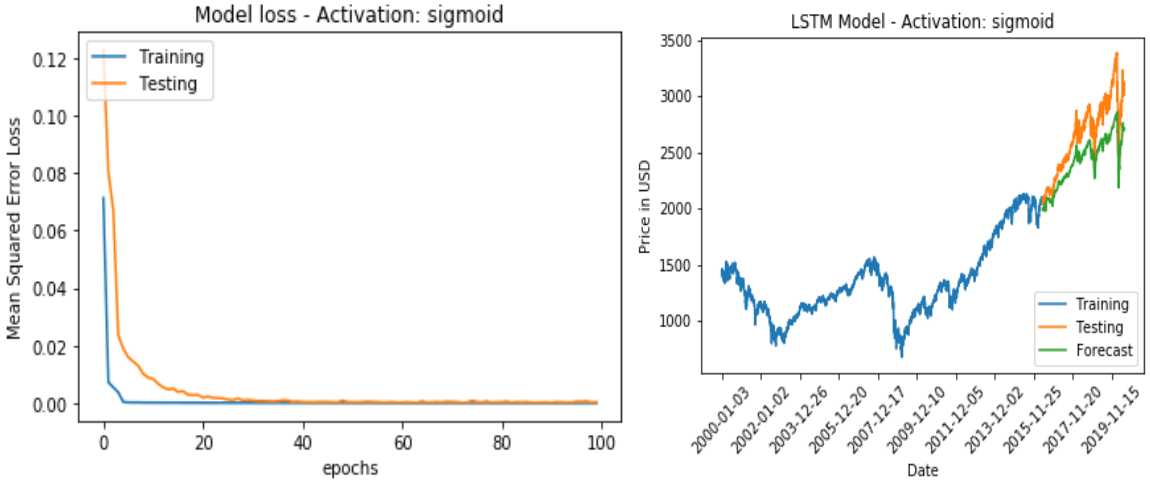
Note: Sensitivity analysis of activation functions in LSTM strategy. The following hyperparameters were used: Training: First 4150 datasets and testing last 1005 datasets of S&P 500 Index. Hyperparameters used: Number of units in hidden layers: 30, Batch size: 64, activation function used for hidden layer: ReLU, loss function: Mean Squared Error, optimizer: Adam, epochs: 100, dropout rate: 0, learning rate: 0.001.

Figure 25. LSTM model with ELU activation functions.



Note: Sensitivity analysis of activation functions in LSTM strategy. The following hyperparameters were used: Training: First 4150 datasets and testing last 1005 datasets of S&P 500 Index. Hyperparameters used: Number of units in hidden layers: 30, Batch size: 64, activation function used for hidden layer: ELU, loss function: Mean Squared Error, optimizer: Adam, epochs: 100, dropout rate: 0, learning rate: 0.001.

Figure 26. LSTM model with sigmoid activation functions.

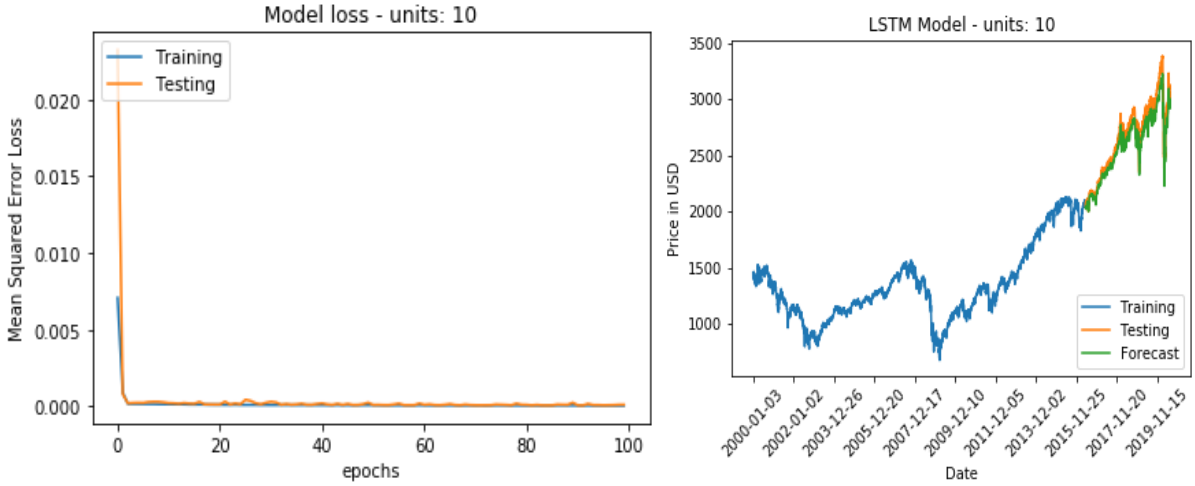


Note: Sensitivity analysis of activation functions in LSTM strategy. The following hyperparameters were used: Training: Note: Training: First 4150 datasets and testing last 1005 datasets of S&P 500 Index. Hyperparameters used: Number of units in hidden layers: 30, Batch size: 64, activation function used for hidden layer: sigmoid, loss function: Mean Squared Error, optimizer: Adam, epochs: 100, dropout rate: 0, learning rate: 0.001.

5.2.5. Number of Units

The number of units in the hidden layer is also known as a node, or sometimes also called a neuron or perceptron. It is a computational unit which has one or multiple weighted input connections, there is a transfer function which combines the inputs in some way, and there is an output connection. Nodes are organized into layers to comprise a neural network. Each of the nodes in the single-layer connects directly to an input variable which then contributes towards an output variable.

Figure 27. LSTM model with the Number of Units equal to 10.

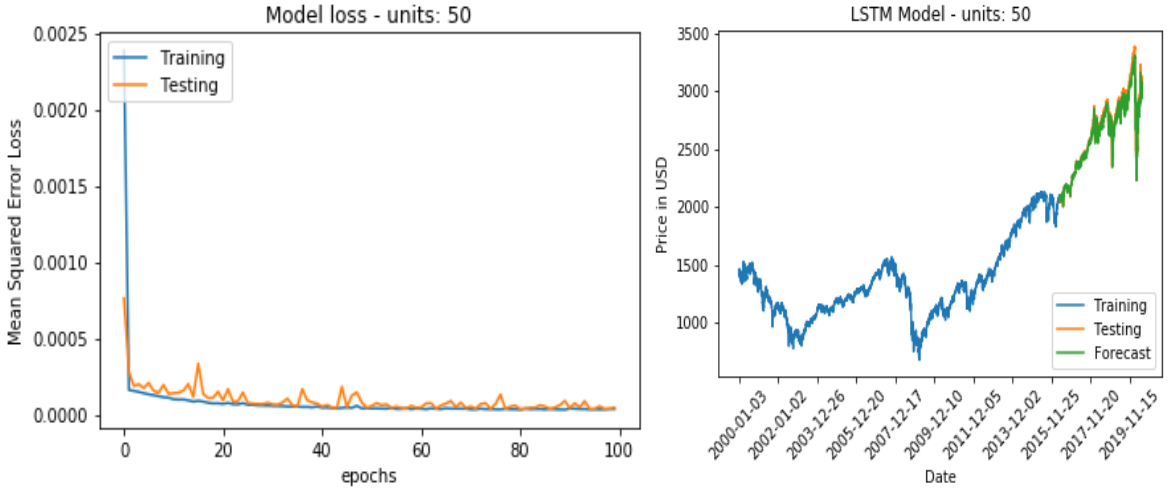


Note: Sensitivity analysis of number of units in LSTM strategy. The following hyperparameters were used: Training: First 4150 datasets and testing last 1005 datasets of S&P 500 Index. Hyperparameters used: Number of units in hidden layers: 10, Batch size: 64, activation function used for hidden layer: tanh, loss function: Mean Squared Error, optimizer: Adam, epochs: 100, dropout rate: 0, learning rate: 0.001.

Next tested hyperparameter is the number of units in the hidden layer. Figure 27, 28, 29, 30 and Table 12 show the results of these changes. The following number of units were tested: 10, 50, 100 and 300. Initially, the number of units was set to be 30 units. It is noted that an increasing number of units in the hidden layer leads to better performance, however increasing

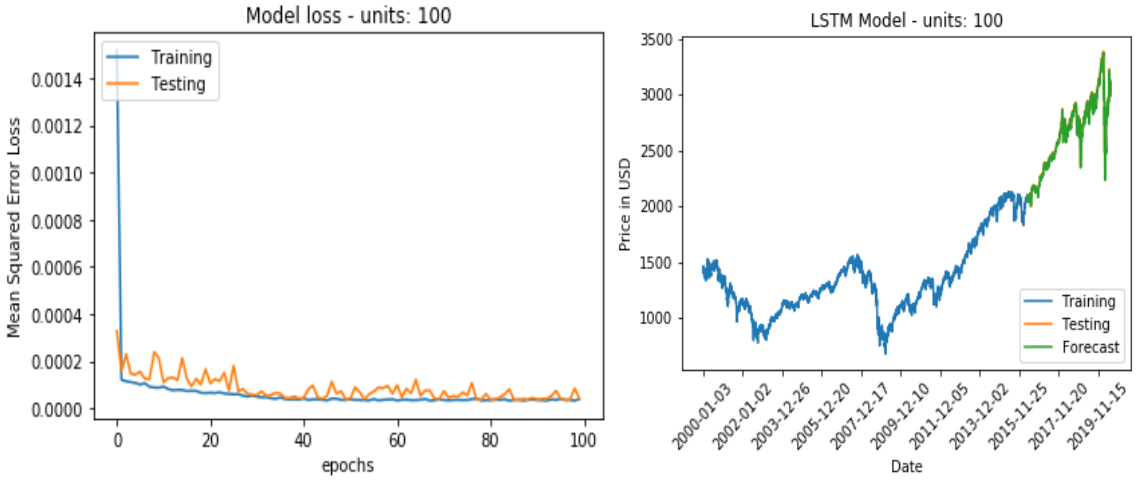
it more than the optimal units worsens the performance, due to the overfitting of the data. However, it depends on the complexity of the problem. It is very hard to find the optimal number of units in the hidden layer. Several search methods such as random search, grid search, heuristic and exhaustive search methods can be used to find an approximation of the optimal number of units to be used in the hidden layer. LSTM with 100 units performed in a similar way as with the initial hyperparameter of 30 units.

Figure 28. LSTM model with the Number of Units equal to 50.



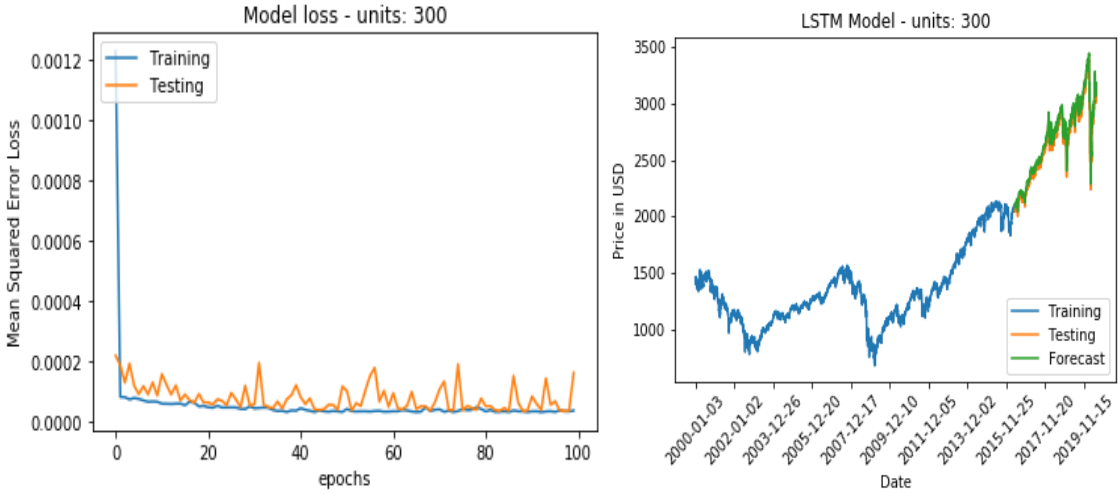
Note: Sensitivity analysis of number of units in LSTM strategy. The following hyperparameters were used: Training: First 4150 datasets and testing last 1005 datasets of S&P 500 Index. Hyperparameters used: Number of units in hidden layers: 50, Batch size: 64, activation function used for hidden layer: tanh, loss function: Mean Squared Error, optimizer: Adam, epochs: 100, dropout rate: 0, learning rate: 0.001.

Figure 29. LSTM model with the Number of Units equal to 100.



Note: Sensitivity analysis of number of units in LSTM strategy. The following hyperparameters were used: Training: First 4150 datasets and testing last 1005 datasets of S&P 500 Index. Hyperparameters used: Number of units in hidden layers: 100, Batch size: 64, activation function used for hidden layer: tanh, loss function: Mean Squared Error, optimizer: Adam, epochs: 100, dropout rate: 0, learning rate: 0.001.

Figure 30. LSTM model with the Number of Units equal to 300.



Note: Sensitivity analysis of number of units in LSTM strategy. The following hyperparameters were used: Training: First 4150 datasets and testing last 1005 datasets of S&P 500 Index. Hyperparameters used: Number of units in hidden layers: 300, Batch size: 64, activation function used for hidden layer: tanh, loss function: Mean Squared Error, optimizer: Adam, epochs: 100, dropout rate: 0, learning rate: 0.001.

Table 12. Error metrics for LSTM Model with various number of units.

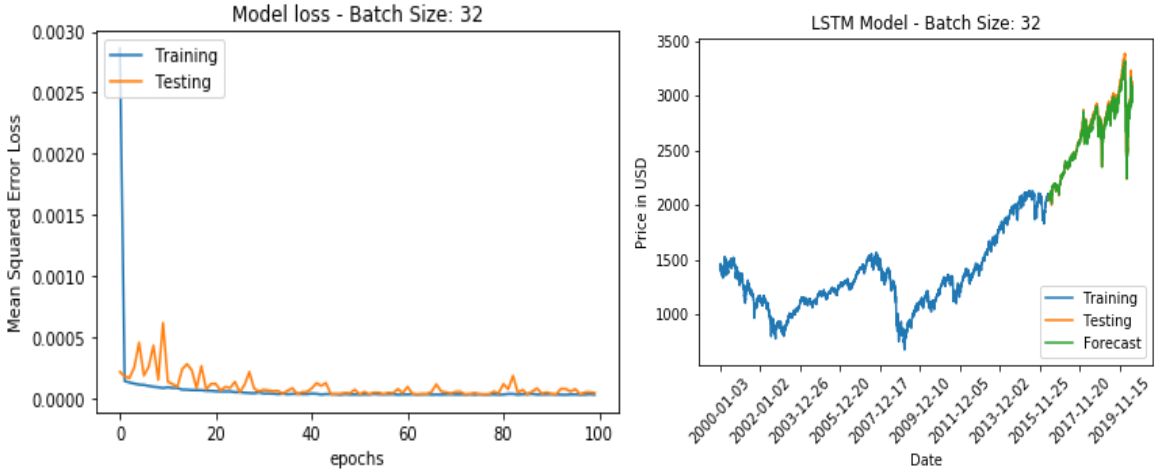
Model Name	MAE	MAPE	MSE	RMSE
Random Walk	958.36	34.42	1135820.64	1066.57
LSTM	21.77	0.81	1399.43	37.40
LSTM: Units:10	73.67	2.77	7079.72	84.14
LSTM: Units:50	32.98	1.20	2220.63	47.12
LSTM: Units:100	**21.45**	**0.80**	**1244.48**	**35.28**
LSTM: Units:300	49.98	1.84	3440.05	58.65

Note: Error metrics for various number of units in LSTM strategy. The following changes in number of units were tested: Number of units in hidden layer: 10, 50, 100 and 300. Initial number of units: 30. Bolded numbers indicate the lowest value of error metric.

5.2.6. Batch Size

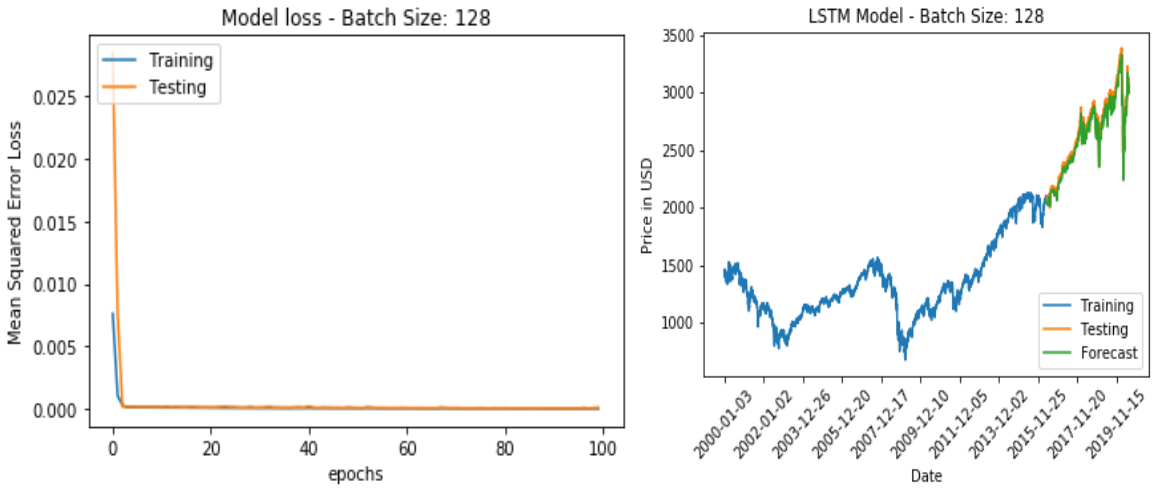
Next tested hyperparameter is the Batch size. Figure 31, 32, 33 and Table 13 show the results of these changes. The following batch sizes were tested: 32, 128 and 256. Batch size has a great impact on learning. It is the number of samples in one iteration. For instance, when a batch is put through the network, it averages the gradients. By taking samples from the dataset, the gradient can be estimated while reducing computational cost significantly. There are of course variants in Gradient Descent Algorithms such as Vanilla Gradient Descent, Stochastic Gradient Descent and Mini-Batch Gradient Descent. The smaller the Mini-Batch, the better would be the performance of the model (not always) and of course it has got to do with the epochs too for faster learning. It is noted that using lower batch sizes with 100 epochs provides better performance. However, none of the batch sizes could beat the 64 set at the beginning.

Figure 31. LSTM model with Batch Size equal to 32.



Note: Sensitivity analysis of batch size in LSTM strategy. The following hyperparameters were used: Training: First 4150 datasets and testing last 1005 datasets of S&P 500 Index. Hyperparameters used: Number of units in hidden layers: 30, Batch size: 32, activation function used for hidden layer: tanh, loss function: Mean Squared Error, optimizer: Adam, epochs: 100, dropout rate: 0, learning rate: 0.001.

Figure 32. LSTM model with Batch Size equal to 128.



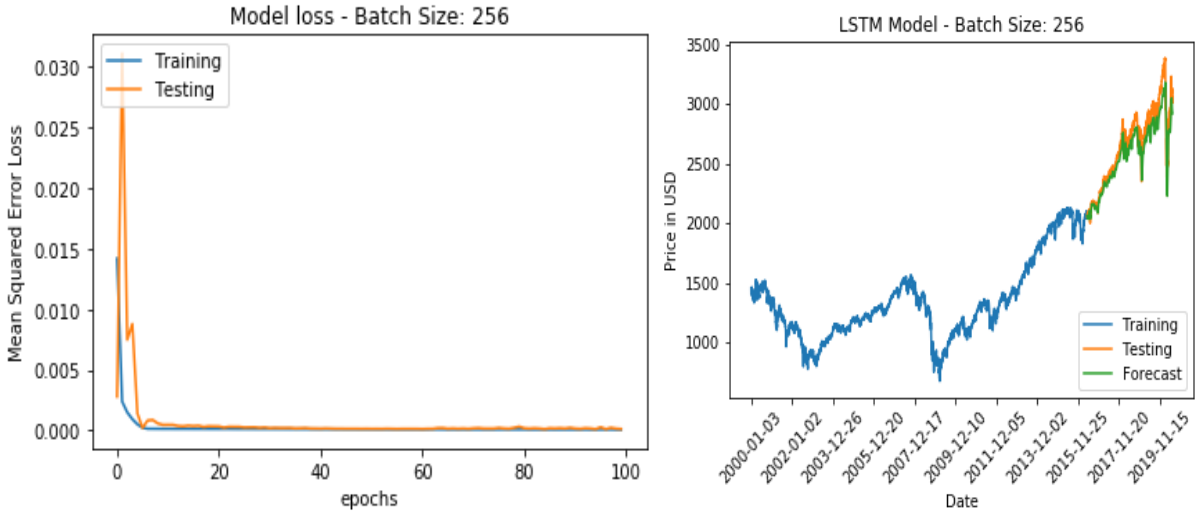
Note: Sensitivity analysis of batch size in LSTM strategy. The following hyperparameters were used: Training: First 4150 datasets and testing last 1005 datasets of S&P 500 Index. Hyperparameters used: Number of units in hidden layers: 30, Batch size: 128, activation function used for hidden layer: tanh, loss function: Mean Squared Error, optimizer: Adam, epochs: 100, dropout rate: 0, learning rate: 0.001.

Table 13. Error metrics for LSTM model with various batch size.

Model Name	MAE	MAPE	MSE	RMSE
Random Walk	958.36	34.42	1135820.64	1066.57
LSTM	**21.77**	**0.81**	**1399.43**	**37.40**
LSTM: Batch size: 32	28.88	1.05	1916.12	43.77
LSTM: Batch size: 128	47.31	1.80	3129.70	55.94
LSTM: Batch size: 256	85.15	3.19	10476.70	102.36

Note: Error metrics for various batch size in LSTM strategy. The following changes in batch size were tested: Batch size: 32, 128 and 256. Initial Batch size: 64. Bolded numbers indicate the lowest value of error metric.

Figure 33. LSTM model with Batch Size equal to 256.

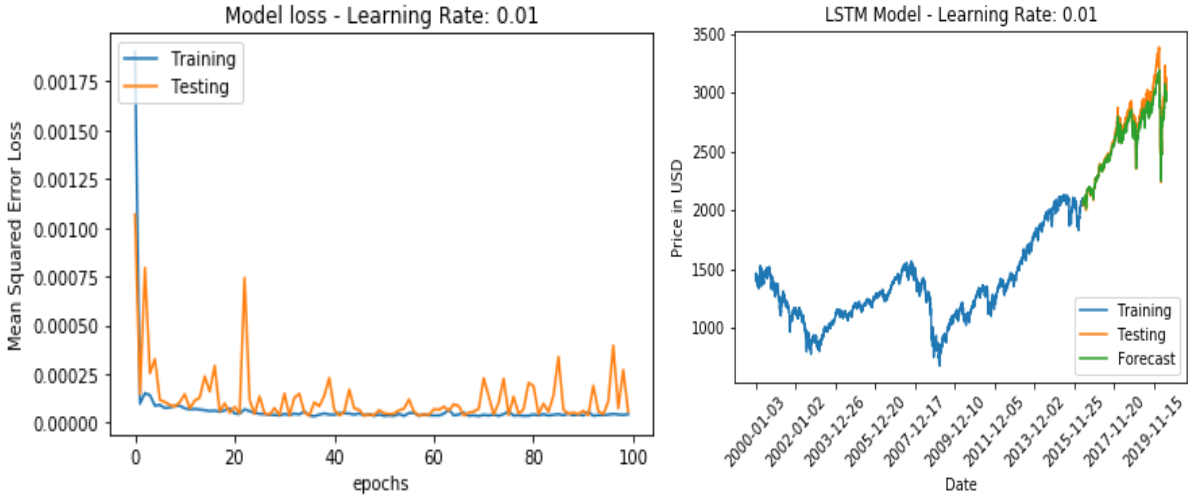


Note: Sensitivity analysis of batch size in LSTM strategy. The following hyperparameters were used: Training: First 4150 datasets and testing last 1005 datasets of S&P 500 Index. Hyperparameters used: Number of units in hidden layers: 30, Batch size: 256, activation function used for hidden layer: tanh, loss function: Mean Squared Error, optimizer: Adam, epochs: 100, dropout rate: 0, learning rate: 0.001.

5.2.7. Learning Rate

Next tested hyperparameter is the learning rate. Figure 34, 35, 36 and Table 14 show the results of these changes. The following values of learning rate were tested: 0.01, 0.0001 and 0.0005. It is noted that lowering the learning rate leads to better performance. The learning rate controls how quick the LSTM model is adapted to the problem. Several optimizers use default learning rates as per its calculation methods. For instance: Adam optimizer uses the default learning rate of 0.001.

Figure 34. LSTM model with learning rate equal to 0.01.

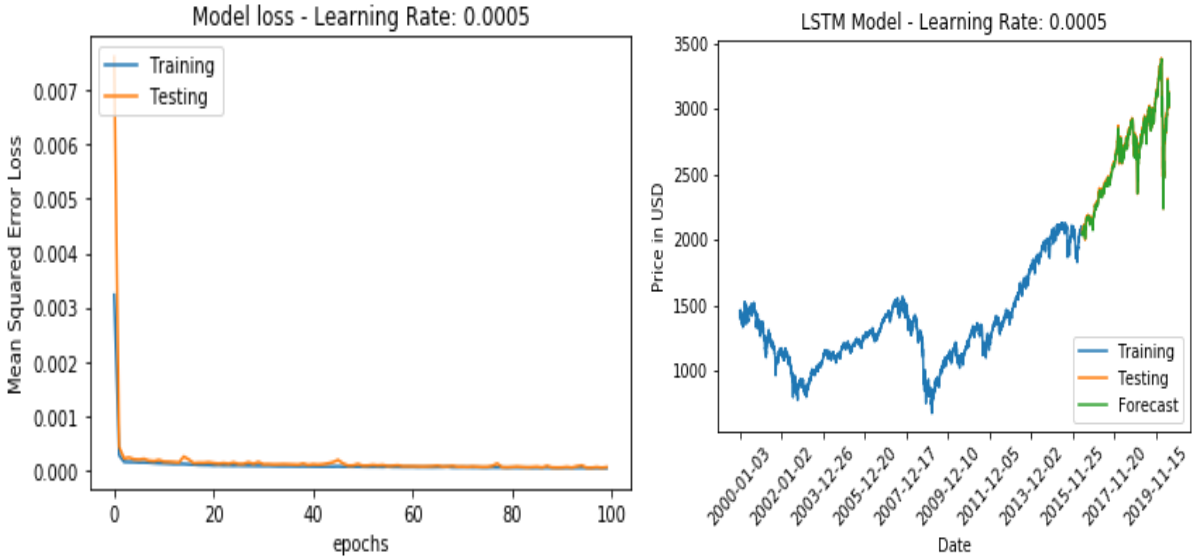


Note: Sensitivity analysis of learning rate in LSTM strategy. The following hyperparameters were used: Training: First 4150 datasets and testing last 1005 datasets of S&P 500 Index. Hyperparameters used: Number of units in hidden layers: 30, Batch size: 64, activation function used for hidden layer: tanh, loss function: Mean Squared Error, optimizer: Adam, epochs: 100, dropout rate: 0, learning rate: 0.01.

It is noted that smaller learning rates require more training epochs considering the smaller changes are made to the weights for each update, whereas larger learning rate results in rapid changes and requires fewer training epochs. A learning rate which is too large can

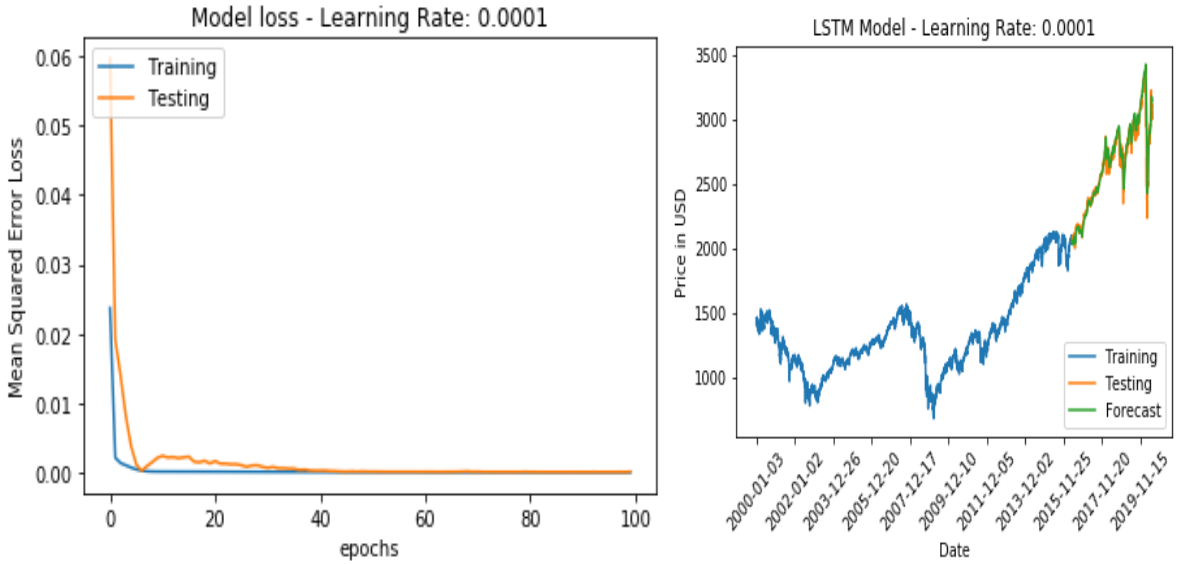
cause the model converging too quickly towards a suboptimal solution, whereas a learning rate which is too small can cause the process to get stuck. The challenge of training deep learning neural networks involves selecting the learning rate in a careful manner as it is one of the most important hyperparameters for the model.

Figure 35. LSTM model with learning rate equal to 0.0005.



Note: Sensitivity analysis of learning rate in LSTM strategy. The following hyperparameters were used: Training: First 4150 datasets and testing last 1005 datasets of S&P 500 Index. Hyperparameters used: Number of units in hidden layers: 30, Batch size: 64, activation function used for hidden layer: tanh, loss function: Mean Squared Error, optimizer: Adam, epochs: 100, dropout rate: 0, learning rate: 0.0005.

Figure 36. LSTM model with learning rate equal to 0.0001.



Note: Sensitivity analysis of learning rate in LSTM strategy. The following hyperparameters were used: Training: First 4150 datasets and testing last 1005 datasets of S&P 500 Index. Hyperparameters used: Number of units in hidden layers: 30, Batch size: 64, activation function used for hidden layer: tanh, loss function: Mean Squared Error, optimizer: Adam, epochs: 100, dropout rate: 0, learning rate: 0.0001.

Table 14. Error metrics for LSTM model with various learning rate.

Model Name	MAE	MAPE	MSE	RMSE
Random Walk	958.36	34.42	1135820.64	1066.57
LSTM	**21.77**	**0.81**	1399.43	37.40
LSTM: Leaning Rate: 0.01	51.27	1.85	4752.21	68.94
LSTM: Leaning Rate: 0.0005	23.09	0.87	**1223.28**	**34.98**
LSTM: Leaning Rate: 0.0001	39.29	1.44	4439.35	66.63

Note: Error metrics for various learning rate in LSTM strategy. The following changes in learning rate were tested: Learning rate: 0.01, 0.0001 and 0.0005. Initial learning rate: 0.001. Bolded numbers indicate the lowest value of error metric.

5.2.8. Summary of LSTM sensitivity analysis

Table 15 summarizes the results of the whole sensitivity analysis for LSTM model.

Table 15. The summary of error metrics for LSTM model.

Model Name	MAE	MAPE	MSE	RMSE
Random Walk	958.36	34.42	1135820.64	1066.57
LSTM	21.77	0.81	1399.43	37.40
LSTM: 50 epochs	37.26	1.37	2453.63	49.53
LSTM: 150 epochs	26.87	0.98	1759.62	41.95
LSTM: 300 epochs	65.95	2.48	5862.99	76.57
LSTM: Loss function: MAE	37.40	1.39	2736.53	52.31
LSTM: Loss function: logcosh	49.42	1.84	3685.27	60.71
LSTM: Loss function: huber loss	29.88	1.10	2020.87	44.95
LSTM: Optimizer: sgd	281.03	11.54	95555.80	309.12
LSTM: Optimizer: RMSprop	105.16	3.98	14158.35	118.99
LSTM: Optimizer: adagrad	184.50	6.27	43970.19	209.69
LSTM: Activation: ReLU	226.45	9.11	58650.27	242.18
LSTM: Activation: ELU	67.27	2.54	5822.19	76.30
LSTM: Activation: sigmoid	230.19	9.21	64842.60	254.64
LSTM: Units:10	73.67	2.77	7079.72	84.14
LSTM: Units:50	32.98	1.20	2220.63	47.12
LSTM: Units:100	**21.45**	**0.80**	1244.48	35.28
LSTM: Units:300	49.98	1.84	3440.05	58.65
LSTM: Batch size: 32	28.88	1.05	1916.12	43.77
LSTM: Batch size: 128	47.31	1.80	3129.70	55.94
LSTM: Batch size: 256	85.15	3.19	10476.70	102.36
LSTM: Leaning Rate: 0.01	51.27	1.85	4752.21	68.94
LSTM: Leaning Rate: 0.0005	23.09	0.87	**1223.28**	**34.98**
LSTM: Leaning Rate: 0.0001	39.29	1.44	4439.35	66.63

Note: Summary of the error metrics for LSTM hyperparameters used during sensitivity analysis. Bolded numbers indicate the lowest value of error metric.

Conclusions

Technology has revolutionized the functions of financial markets and the way the financial assets are traded. Two significant technological changes are that the investors are using computers to automate their trading activity and the markets rearranging themselves in such a way that virtually all markets are now based on electronic limit order books (Jain, 2005). The quality and speed of access to such markets encourage the use of algorithmic trading by investors. What is more, the main aim of this automated trading was to delete human errors in

trading and risk management. None of the trading strategies could provide perfect forecasting methods, however, the investors are always interested in trading strategies that are able to predict the future price with high accuracy. The main aim of this paper is to investigate which forecasting methods provide the best predictions with regards to lower forecasting errors. A classical model such as ARIMA is compared with a machine learning model such as LSTM.

This research focuses on the performance of ARIMA and LSTM using various error metrics. The study assumes that there is asymmetric information in the stock market which makes it possible to analyze the volatility in the stock prices, as well as predicting the prices within available historical data. This paper uses S&P 500 index close prices from 03.01.2000 t0 29.06.2020. This study uses Random Walk model generating naïve forecasts as a benchmark model. Both ARIMA and LSTM perform better than the benchmark model, in terms of lower forecasting errors.

The results for all the models were obtained using Python (version 3.8.5) with Jupyter Notebook (version 6.0.3) development environment. All the relevant libraries and packages were installed using Anaconda Prompt (conda version 4.8.5) on a 64-bit Windows 10 operating system. Deep learning libraries used for designing, training, and testing the neural network are Keras (version 2.4.3) and TensorFlow (version 1.8.0).

At the beginning of this research four main hypotheses and one research question were stated, and they were verified and answered during the empirical part. The first hypothesis is rejected since the result showed that ARIMA outperforms LSTM in terms of one-step ahead forecasts, which is similar to the results obtained by (Siami-Namini et al., 2018). Several combinations of hyperparameters for LSTM were tested during sensitivity analysis sections, however, it was unable to beat the ARIMA (1,1,1). The second hypothesis that *LSTM is robust to changes in the hyperparameters* was rejected since the results obtained during sensitivity analysis do not show robustness of this model. The third hypothesis, i.e. *Increasing the number of epochs leads to better performance of the model, however using more than the optimal number of epochs leads to an over-fitted model* was not rejected based on the sensitivity analysis. The fourth one, i.e. U*sing the wrong optimizer and activation function can significantly worsen the accuracy of the LSTM model*, was not rejected based on our results. Finally, research question (RQ1): *Can we use one step ahead forecasts from ARIMA or LSTM model in buy/sell signals of investment strategies?* can be answered positively based on the results of this study.

This study can be extended by testing more hyperparameters of the LSTM model and using rolling regression for forecasting. The use of cross-validation such as K-fold, Stratified K-Fold, and Leave-P-Out can be applied instead of one step forecasting. More various strategies can be implemented and combining them into a complex sophisticated model could provide better performance. Additionally, investment could be done based on the models, where buy/sell/hold signal generations could be implemented and then performance statistics such as Information Ratio, Maximum Drawdown could be calculated, which would be more attractive to the investors. As for LSTM, looking for optimization of hyperparameters using grid search could be implemented in future works.

References

A. H. Bukhari, M. A. Z. Raja, M. Sulaiman, S. Islam, M. Shoaib and P. Kumam, 2020, *Fractional Neuro-Sequential ARFIMA-LSTM for Financial Market Forecasting*, in *IEEE Access*, vol. 8, pp. 71326-71338, doi: 10.1109/ACCESS.2020.2985763.

Adil Moghar, Mhamed Hamiche, 2020, *Stock Market Prediction Using LSTM Recurrent Neural Network*, Procedia Computer Science, Volume 170, Pages 1168-1173, ISSN 1877-0509, https://doi.org/10.1016/j.procs.2020.03.049.

Akaike, 1974, A *new look at the statistical model identification*. IEEE Trans. Automat. Control. vAC-19. 716-723

Baek, Y., & Kim, H., 2018. *ModAugNet: A new forecasting framework for stock market index value with an overfitting prevention LSTM module and a prediction LSTM module.* Expert Systems With Applications, 113, 457-480. https://doi.org/10.1016/j.eswa.2018.07.019

Bengio, Y., Simard, P., & Frasconi, P., 1994, *Learning long-term dependencies with gradient descent is difficult.* IEEE Transactions On Neural Networks, *5*(2), 157-166. https://doi.org/10.1109/72.279181

Box, G. and Jenkins, G., 1970, *Time Series Analysis: Forecasting and Control*. Holden-Day, San Francisco.

Brogaard, Jonathan and Hendershott, Terrence J. and Riordan, Ryan, 2016, *High Frequency Trading and the 2008 Short Sale Ban (March 30, 2016)*. Journal of Financial Economics (JFE), Forthcoming, Available at SSRN: https://ssrn.com/abstract=2509376 or http://dx.doi.org/10.2139/ssrn.2509376

Chiu C.C., Cook D.F., Pignatiello J.J., 1995, *Radial basis function neural network for Kraft pulping forecasting*. International Journal of Industrial Engineering 2(2): 209-215

Chlebus M., DyczkoM., Woźniak M., 2020, *Nvidia's stock returns prediction using machine learning techniques for time series forecasting problem*, Working Papers of Faculty of Economic Sciences, University of Warsaw, WP 22/2020 (328), https://www.wne.uw.edu.pl/files/6415/9481/5844/WNE_WP328.pdf

Cook D.F., Chiu C.C., 1997, *Predicting the internal bond strength of particleboard utilizing a radial basis function neural network*, Engineering Applications AI 10(2): 171-177

Cowpertwait, P., & Metcalfe, A., 2009, *Introductory time series with R* (p. 142). Springer.

F. Qian and X. Chen, 2019, *Stock Prediction Based on LSTM under Different Stability*, 2019 IEEE 4th International Conference on Cloud Computing and Big Data Analysis (ICCCBDA), Chengdu, China, pp. 483-486, doi: 10.1109/ICCCBDA.2019.8725709.

Gao X.M., Gao X.Z., Tanskanen J., Ovaska, S.J., 1997, *Power prediction in mobile communications systems using an optimal neural network structure*. IEEE Transportation Neural Networks 8(6), 1446-1455

Hannan, E.J. and Quin, G.G.,1979, *The determination of the order of an autoregression*. J.R. Statistic. Soc. B, 41, 190-195.

Hochreiter, S., & Schmidhuber, J., 1997, *Long Short-Term Memory*. *Neural Computation, 9(8),* 1735-1780. https://doi.org/10.1162/neco.1997.9.8.1735s

Jain, P., 2005, *Financial market design and the equity premium: Electronic versus floor trading*. Journal of Finance 60 (6), 2955–2985.

Jing Zhang, Shicheng Cui, Yan Xu, Qianmu Li, and Tao Li, 2018. *A Novel Data-driven Stock Price Trend Prediction System*, Elsevier Expert Systems with Applications, Vol. 97, pp.60–69.

Jonathan Creighton, and Farhana H Zulkernine, 2017, *Towards Building a Hybrid Model for Predicting Stock Indexes*, IEEE International Conference on Big Data.

K Hiba Sadia, Aditya Sharma, Adarrsh Paul, Sarmistha Padhi, and Saurav Sanyal, 2019. *Stock Market Prediction Using Machine Learning Algorithms*, International Journal of Engineering and Advanced Technology, Vol. 8, No. 4, pp.25-31.

Kijewski M., Ślepaczuk R., 2020, *Predicting prices of S&P500 index using classical methods and recurrent neural networks*. Working Papers of Faculty of Economic Sciences, University of Warsaw, WP 27/2020 (333), https://www.wne.uw.edu.pl/files/6215/9765/7140/WNE_WP333.pdf

Kirilenko, Andrei A. and Lo, Andrew W., 2013, *Moore's Law vs. Murphy's Law: Algorithmic Trading and Its Discontents*. Journal of Economic Perspectives, Available at SSRN: https://ssrn.com/abstract=2235963 or http://dx.doi.org/10.2139/ssrn.2235963

Liu, C.D., Wang, J.H., Xiao, D. and Liang, Q., 2016, *Forecasting S&P 500 Stock Index Using Statistical Learning Models.* Open Journal of Statistics, 6, 1067-1075. http://dx.doi.org/10.4236/ojs.2016.66086

Luca Di Persio, and Oleksandr Honchar, 2017, *Recurrent Neural Networks Approach to the Financial Forecast of Google Assets*, International Journal of Mathematics and Computers in Simulation, Vol. 11, pp.7-13.

Makridakis, S., Wheelwright, S., and McGee, V., 1983, *Forecasting Methods and Applications (2nd Edition)*, Wiley and Sons, New York.

Mehtab, S., & Sen, J., 2019. *A Robust Predictive Model for Stock Price Prediction Using Deep Learning and Natural Language Processing*. SSRN Electronic Journal. https://doi.org/10.2139/ssrn.3502624

Min Wen, Ping Li, Lingfei Zhang, and Yan Chen, 2019, *Stock Market Trend Prediction Using High-Order Information of Time Series*, IEEE Access, Vol. 7, pp.28299-28308.

Mohammad Asiful Hossain, Rezaul Karim, Ruppa Thulasiram, Neil D B Bruce, and Yang Wang, 2018. *Hybrid Deep Learning Model for Stock Price Prediction*, IEEE Symposium Series on Computational Intelligence SSCI, pp.1837-1844

Patterson, J. and Gibson, A., 2017, *Deep Learning: A Practitioner's Approach*. O'Reilly Media, Inc., Sebastopol.

Pindyck, R.S., Rubinfeld D.L. 1998. *Econometric Models and Economic Forecasts*. 4th edn. McGraw-Hill International Editions

Raut Sushrut Deepak, Shinde Isha Uday, and Dr D Malathi, 2017, *Machine Learning Approach in Stock Market Prediction*, International Journal of Pure and Applied Mathematics, Vol. 115, No. 8, pp.71-77.

Reilly D.P., 1980, *Experiences with an automatic Box-Jenkins modelling algorithm. Time Series Analysis-Proceedings of Houston Meeting on Time Series Analysis*. Amsterdam, North-Holland Publishing

Reynolds S.B., Mellichamp J.M., Smith R.E., 1995, *Box-Jenkins forecast model identification. AI Expert*, June, 15-28

Saad E.W., Prokhorov D.V., Wunsch D.C., 1998, *Comparative study of stock trend prediction using time delay, recurrent and probabilistic neural networks*. IEEE Transportation Neural Networks 9(6): 1456-1469

Sahaj Singh Maini, and Govinda K, 2017. *Stock Market Prediction using Data Mining Techniques*, Proc. Int. Conf. Intelligent Sustainable Systems, IEEE Xplore Compliant - Part Number: CFP17M19-ART, pp.654-661.

Sakshi, K., & A, V. , 2020. *An ARIMA- LSTM Hybrid Model for Stock Market Prediction Using Live Data*. Journal Of Engineering Science And Technology Review, *13*(4), 117-123. https://doi.org/10.25103/jestr.134.11

Schwartz E.S, 1997, *The Stochastic Behavior of Commodity Prices: Implications for Valuation and Hedging*. J Finance 52(3) Papers and Proceedings Fifty-Seventh Annual Meeting, American Finance Association, New Orleans, Louisiana January 4-6, (July 1997), 923-973.

Siami-Namini, S., Tavakoli, N., & Siami Namin, A., 2018, *A Comparison of ARIMA and LSTM in Forecasting Time Series*. 2018 17Th IEEE International Conference on Machine Learning and Applications (ICMLA). doi: 10.1109/icmla.2018.00227

Sreelekshmy Selvin, Vinayakumar R, Gopalakrishnan EA, Vijay Krishna Menon, and Soman KP, 2017. *Stock Price Prediction using LSTM, RNN and CNN- Sliding Window Model*, IEEE Conference Paper, doi: 10.1109/ICACCI.2017.8126078, pp.1643- 1647

Yang Y., Zheng Y., Hospedales T., 2017, *Gated neural networks for option pricing: rationality by design*, Association for the Advancement of Artificial Intelligence.

Zenkova M., Ślepaczuk R., 2018, *Robustness of Support Vector Machines in Algorithmic Trading on Cryptocurrency Market*, Central European Economic Journal, 5(1), pp. 186-205.

University of Warsaw
Faculty of Economic Sciences
44/50 Długa St.
00-241 Warsaw
www.wne.uw.edu.pl