Electronic Research Journal of Engineering, Computer and Applied Sciences
www.erjsciences.info Volume 2 (2020)

Predicting the Stock Market using Machine Learning: Long short-term Memory

Khalid Bin Saboor[1]
Department of Finance, SEM, Beijing University of Aeronautics & Astronautics, Beijing, P.R. China
E-mail: khalidsaboor@buaa.edu.cn

Qurat Ul Ain Saboor
Faculty of Contemporary Studies, National Defence University, Islamabad, Pakistan

Liyan Han
Department of Finance, SEM, Beijing University of Aeronautics & Astronautics, Beijing, P.R. China

Abdul Saboor Zahid
Faculty of Contemporary Studies, National Defence University, Islamabad, Pakistan

Abstract: The stock market is notorious for its intense uncertainty and instability, and researchers and investors alike often try a detailed and useful way to direct their stock trading. Long short-term memory (LSTM) neural networks are a subtype of Recurrent neural networks (RNNs) having significant practical utility in a wide variety of applications. Moreover, due to its unique ability to 'remember,' LSTMs do not depend on the long-term and can, therefore help forecast financial time series such as the stock market. In this study, we use sci-kit learn's min-max scaler to transform the data, extract features, and establish our model for prediction. To make our analysis holistic, we use daily price data for two entities listed on two different stock exchanges. All stages of the study have been conducted using various libraries of the Python programming language using the iPython Notebook. Our results suggest that LSTMs may be more effective than traditional linear techniques such as ARIMA since the latter can not capture the non-linear factors of a problem. Furthermore, even though LSTM is better for the issue at hand, they may perform worse for others unless tuned accordingly.

Keywords: Python, AI, LSTM, Stock Market, etc.

Introduction:

The problem of forecasting the stock market, although a tricky one, has still been given great focus by investors and researchers alike (Dingli and Fournier, 2017). The stock market, by its nature, is noisy and random, and is thus very hard to predict (Reddy, and J, 2019). Keeping this in mind, it would not make sense to use only one's experience and traditional expertise of trading in said market. There is a need to involve something new, which would be not only intelligent but also valid for the said problem. This is why, with ever-increasing computing power and easy access to scientific learning, new artificial intelligence and machine learning techniques have been researched and thus become a significant focus for the people involved in the stock market. Neural

[1] Corresponding author

networks have an inherent ability that enables them to capture nonlinear aspects of a problem, and thus they are an excellent predictor. Long short-term memory (LSTM) neural networks have proven to be effective in other issues such as text processing, speech recognition, and computer vision (Cai and Liu, 2016; Girirajan et al., 2020) among others (Tang et al., 2019; Uchida et al., 2008). Also, since LSTMs possess the ability to memorize, they can be beneficial for non-stationary time series such as a stock market. They should, in theory, at least, be more effective than traditional statistical methods, again, since stock data is noisy, non-stationary, and non-linear (Pavlyuchenko, 2019). This is not to say that AI techniques are perfect at tackling the said problem. However, there is still hope that an 'educated guess' can help address it (Meharia, 2011).

The literature shows that throughout time, researchers have been attempting to do just the same. Erdinc (2005) conducted a comparative study of ANNs vs. linear regression for the stock market. They concluded that ANNs were indeed better for the said problem. In 2014, Chauhan et al. also used ANNs for the same problem and found that it increased the predictive ability over linear techniques (Chauhan et al., 2014). In the same year, Wanjawa et al. (2014) predicted stock prices that used a feed-forward MLP for the problem. Their results were also positive. Taking a more complicated approach, Wang et al., (2012) proposed a hybrid prediction algorithm that combined ESM, ARIMA, and backpropagation neural networks for the task. They conducted a holistic analysis that concluded that the synergy attained from combining both linear and non-linear models helped outperform both individually. Models using LSTM and RNN were implemented by Jiang et al., (2020). They also concluded that LSTMs was a good fit for a stock market problem. More recently, last year, Kim and Kim (2019) proposed the feature fusion long short-term memory-convolutional neural network (LSTM-CNN) model, and also concluded that this hybrid outperformed the individuals.

Sharaf and Lie (1995) put forward a model based on the short term involving fuzzy logic neural networks. A Grey-Markov forecasting model was proposed by Wang and Meng (2008) for the purpose of predicting Chinese electricity demand. An adaptive neural-wavelet model, also for the short term, was developed by Zhang and Dong (2001) for the same purpose. Maia, Gonclaves, and El-Keib also proposed hybrid forecasting models for electricity prediction problems (Maia and Gonçalves, 2008; El-Keib et al., 1995).

The Algorithm:

LSTMs are a subtype of RNN (recurrent neural network) that have recently seen their popularity rise (LeCun et al., 2015). LSTM networks aim to solve the long term prediction problem and are thus suitable for time series problems. LSTMs were first introduced by Hochreiter and Schmidhuber (1997). They are different from traditional RNNs in that they possess some contextual state cells that perform as long or short-term memory cells. The output of the LSTMs depends on what state these cells are in. This helps to get a historical context instead of a single look back.

LSTMs retain or 'memorize' information by way of a folded loop that allows it to move in between steps. Thus, much like a human brain, LSTMs predict using past information from gained context. Fig 1 shows an unfolded loop. It shows the folded form of a part of an LSTM neural network where *A* denotes that part of the NN, which is taking in some input $\boldsymbol{X_t}$ outputting some value $\boldsymbol{h_t}$.

The second part shows the same part of the network unfolded. Like RNNs, LSTMs also have a chain-like structure, with a single difference: the repetition module. Also, they have four NN layers instead of only one like in the RNN. Fig 2 demonstrates these structures.

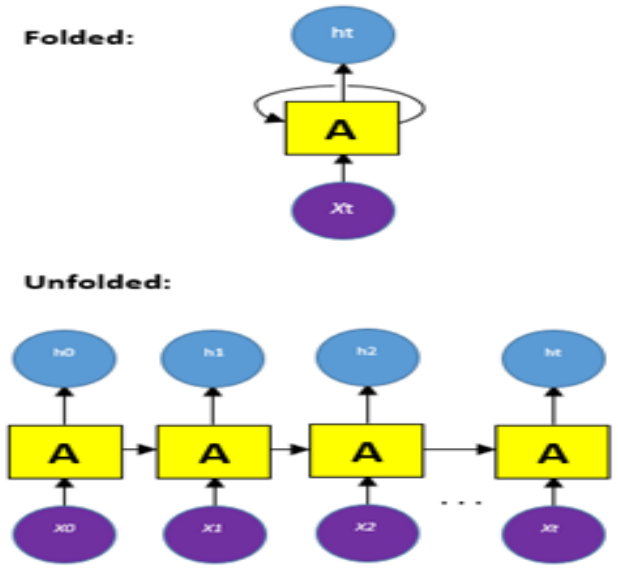


Fig1: Unfolded Loop[2]

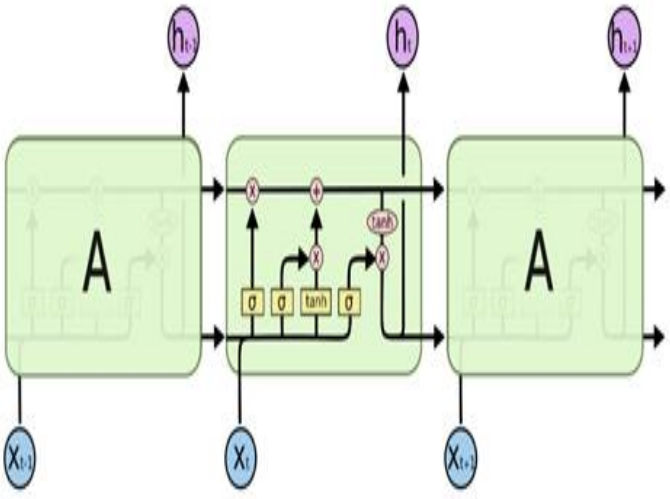


Fig 2: LSTM structure[3]

LSTMs work by filtering information through their cells. This maintains and updates the states of the memory cells. LSTMs consists of four 'gates,' namely, input, forget, and output. Each

[2] Source: Word drawn by Authors
[3] Source: colah.github.io

of the memory cells contains a total of 4 layers. One of these is a *tanh* later, while the others are sigmoid layers.

The forget gate is critical since it determines what information to retain and what to forget. As is visible in Fig 2 (NN part), the current information is denoted by $\boldsymbol{x_t}$, while previous information is indicated by $\boldsymbol{h_{t-1}}$. The memory cell takes the output of these as inputs and combines them as a vector denoted by $[\boldsymbol{h_{t-1}}, \boldsymbol{x_t}]$. The combinations go through a sigmoid transformation and become:

$$\boldsymbol{f_t} = \boldsymbol{\sigma}(\boldsymbol{W_f} . [\boldsymbol{h_{t-1}}, \boldsymbol{x_t}] + \boldsymbol{b_f}) \quad (1)$$

Where:

$\boldsymbol{W_f}$ = The weight matrix,

$\boldsymbol{b_f}$ = The bias, and

$\boldsymbol{\sigma}$ = The sigmoid functional.

Mainly, the forget gate's task is to determine how much the cell state of $\boldsymbol{C_{t-1}}$ is reserved with the cell state of $\boldsymbol{C_t}$. The gate, based on the values of $\boldsymbol{h_{t-1}}$ and $\boldsymbol{x_t}$, will put out a value that is between 0 and 1, with 0 denoting no reservation while 1 indicating the polar opposite.

The input gate's task is to decide how much of $\boldsymbol{x_t}$ is taken into the cell state $\boldsymbol{C_t}$, thereby preventing insignificant information from entering. Furthermore, it has two functions. One, to find the value to be updated using the sigmoid layer (Eq 2), and second, to update the information. Simply put, it needs to see who to give information to and what information needs to be provided. The *tanh* layer creates a new vector $\widehat{\boldsymbol{C}}_{\boldsymbol{t}}$, that controls how much information to add (Eq 3). Finally, the state of the cells is updated (Eq 4).

$$\boldsymbol{i_t} = \boldsymbol{\sigma}(\boldsymbol{W_f} . [\boldsymbol{h_{t-1}}, \boldsymbol{x_t}] + \boldsymbol{b_i}) \quad (2)$$

$$\widehat{\boldsymbol{C}}_{\boldsymbol{t}} = \boldsymbol{tanh}(\boldsymbol{W_c} . [\boldsymbol{h_{t-1}}, \boldsymbol{x_t}] + \boldsymbol{b_c}) \quad (3)$$

$$\boldsymbol{C_t} = \boldsymbol{f_t} * \boldsymbol{C_{t-1}} + \boldsymbol{i_t} * \widehat{\boldsymbol{C}}_{\boldsymbol{t}} \quad (4)$$

Looking at the output gate, this gate is in charge of how much of the current cell's state is let go of. This process is shown in Eq.5.

$$\boldsymbol{O_t} = \boldsymbol{\sigma}(\boldsymbol{W_\sigma} . [\boldsymbol{h_{t-1}}, \boldsymbol{x_t}] + \boldsymbol{b_o}) \quad (5)$$

The process ends when the final value of the cell is output as follows:

$$\boldsymbol{h_t} = \boldsymbol{O_t} * \boldsymbol{tanh}(\boldsymbol{C_t}) \quad (6)$$

The Predictive Model:

Establishing any stock prediction model involves four stages, namely, the collection of data, preprocessing, training, and finally, evaluation. Fig 8 shows a flow chart of the implementation. Fig 3 shows the structure of the LSTM model (a sequential LSTM model),

consisting of an input layer, a hidden layer, and an output layer. The hidden layer comprises two LSTM layers, each with 50 units and a single unit dense layer.

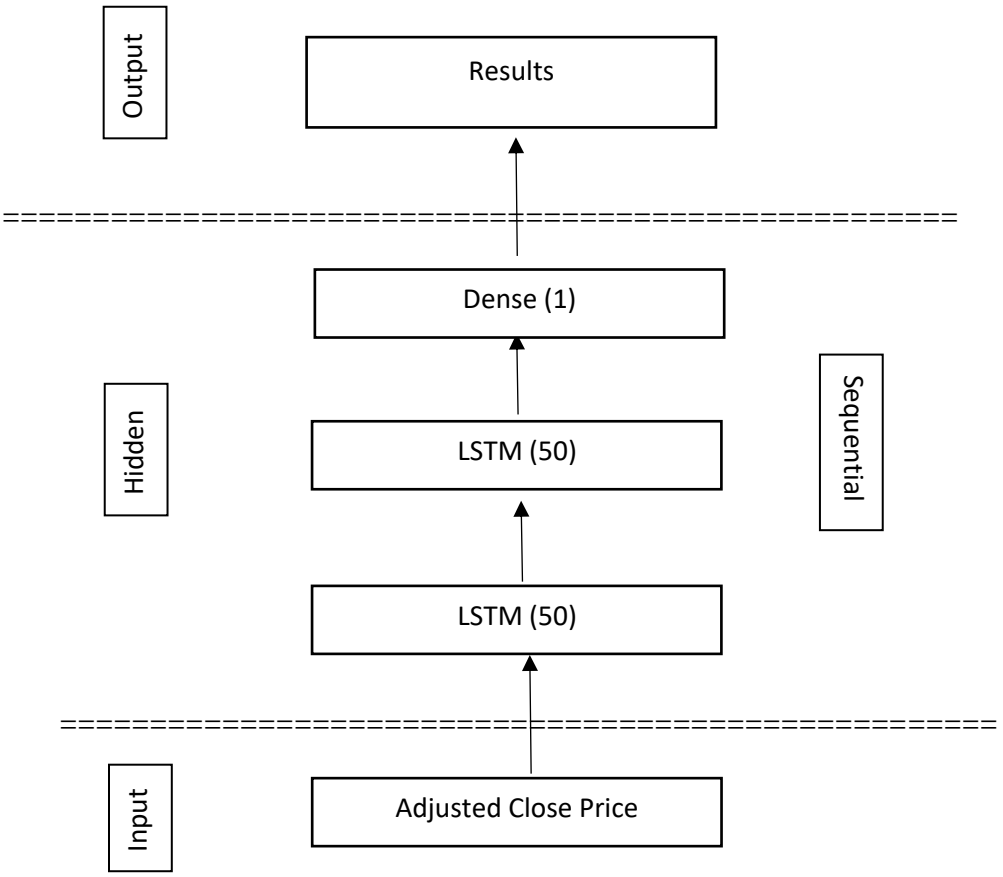
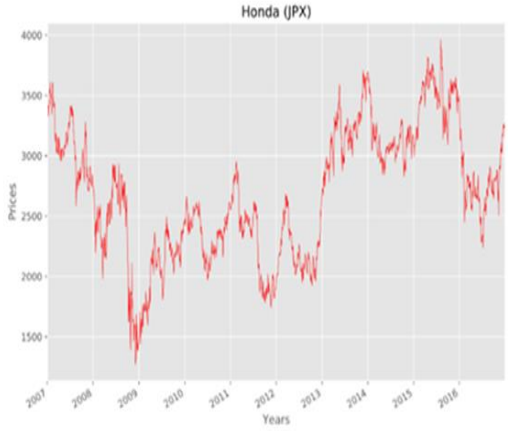


Fig 3: Struture of the LSTM[4]

Data:

Two entities on two different stock exchanges were selected. The first is Honda Motor Co., listed on the TSE. The second is Lenovo Group Limited, registered on the HKG. The snapshots and adjusted closing price graphs of both are shown below:

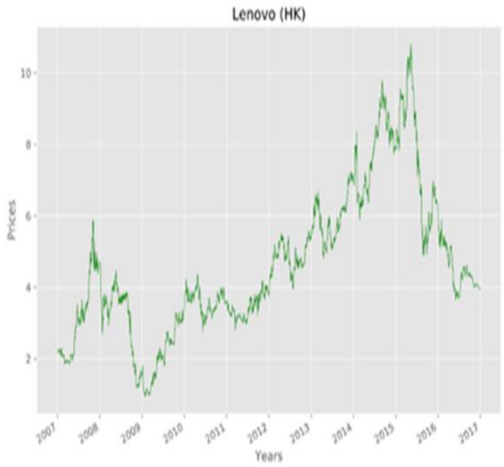
[4] Source: Word drawn by Authors




```
In [17]: print(series['HNDA'].head(5))
         print('Total items:',len(series['HNDA']))

Date
2007-01-04    3443.083740
2007-01-05    3341.388184
2007-01-09    3363.180420
2007-01-10    3276.013184
2007-01-11    3290.541992
Name: HNDA, dtype: float64
Total items: 2120
```

Fig 4: Honda Snapshot[5]




```
In [18]: print(series['LVOG'].head(5))
         print('Total items:',len(series['LVOG']))

Date
2007-01-04    2.185924
2007-01-05    2.165931
2007-01-09    2.279226
2007-01-10    2.219246
2007-01-11    2.185924
Name: LVOG, dtype: float64
Total items: 2120
```

Fig 5: Lenovo Snapshot[6]

These data were downloaded directly to CSV files in Python from Yahoo Finance using the Fetcher module included in the Yahoo_Historical library (Fig 6). As can be seen, from the data, the dates in both have been aligned so that both have an equal number of items, i.e., 2120. This, however, will change during further preprocessing when null items from the data are removed. This is the next step, along with splitting the data into train and test sets at 75%. This is shown in Fig 7.

[5] Snapshot Source: Authors' python codes and output
[6] Source: Authors' python codes and output

```
In [4]: ##data download and export to CSV files for later use
from yahoo_historical import Fetcher

#Honda JPX
HNDA = Fetcher("7267.T", [2007,1,1], [2017,1,1])
HNDAD=(HNDA.getHistorical())
HNDAD.to_csv('HNDA.csv')

#Lenovo Group HK
LVOG = Fetcher("0992.HK", [2007,1,1], [2017,1,1])
LVOGD=(LVOG.getHistorical())
LVOGD.to_csv('LVOG.csv')
```

Fig 6: Fetcher module for data[7]

```
In [24]: #Data Length : Honda
print(len(data))
Tlen= len(data)*.75
print(Tlen)
print(len(data)-Tlen)

2701
2025.75
675.25

#splitting the dataset
#Honda
lstm = new_data.values
train = lstm[0:2026,:]
valid = lstm[2026:,:]

In [58]: #Data Length : Lenovo
print(len(data))
Tlen= len(data)*.75
print(Tlen)
print(len(data)-Tlen)

2716
2037.0
679.0

#splitting the dataset
#Lenovo
lstm = new_data.values
train = lstm[0:2037,:]
valid = lstm[2037:,:]
```

Fig 7: Train & Test split[8]

The experiments were designed, written, and run on a powerful machine with the following specifications:

Hardware:

Intel Core i7 5820K (4.3 OC), 16 GiB Memory

Nvidia GTX770 2GiB – Compute Capability 3.0

RAID0 Storage

Memory on Quad Channel & XMP 2.0

[7] Source: Authors' python codes
[8] Source: Authors' python codes and output

Software:

Windows 10 Pro x64 (Genuine)

Python v3.7.6 (TF-GPU v2.0.0 – for ML)

CUDnn v9.1 (for ML)

Jupyter Notebook v 6.0.3

PyCharm Community

All software above is either open source or allowed for free for academic use, except for MS Windows, which is a genuine copy.

Experiment:

Model Performance Metrics:

We evaluated the performance of the model, i.e., prediction results using Mean Squared Error, Root Mean Squared Error and Mean Absolute Error. These are as follows:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\widehat{Y}_i - Y_i) \quad (7)$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\widehat{Y}_i - Y_i)} \quad (8)$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|\widehat{Y}_i - Y_i| \quad (9)$$

Where:

n = the number of samples,
$\widehat{Y}_i$ = the value predicted, and
Y_i = the real value.

Taking it in statistical terms, the mean squared error (MSE) or Mean Squared Deviation (MSD) is capable of measuring the ways of the squares of observed errors. This is the mean difference (squared) between the predicted or estimated values and the actual observed values. This measure is a function that measures risk, and that corresponds to the squared error loss's expected value.

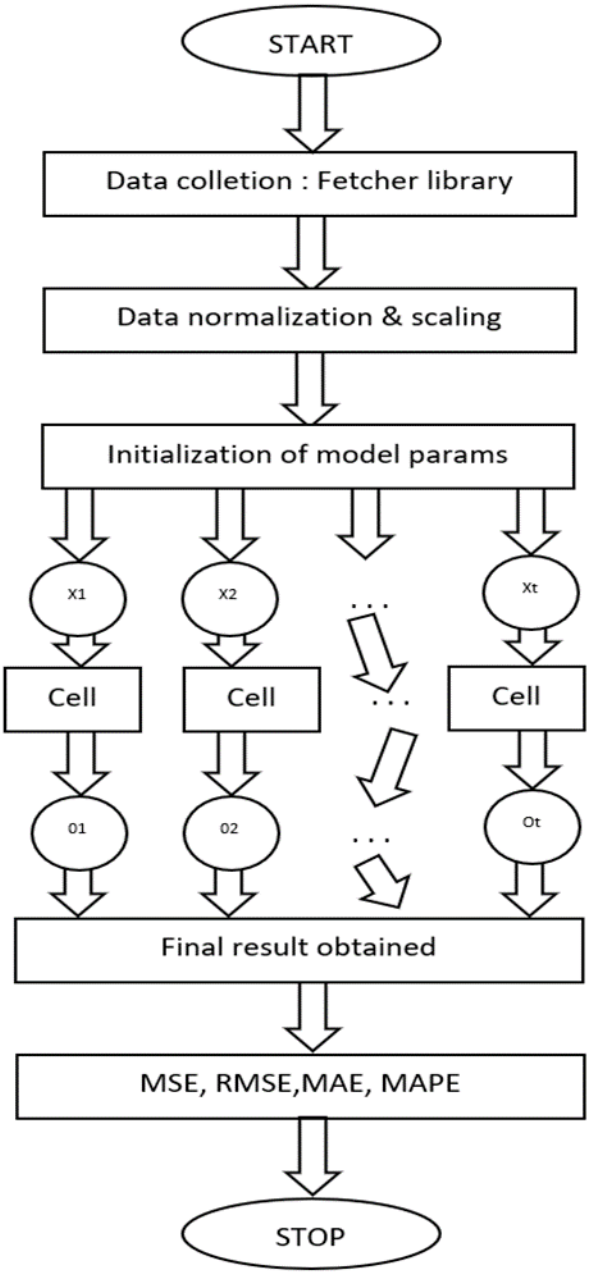


Fig 8: Model flowchart[9]

The MSE value will almost always be a positive and non-zero number. This may be because of its random input nature or perhaps because there are some features of the information that are capable of producing a more accurate forecast, but are not accounted for by MSE (Lee and Priebe, 2000).

[9] Source: Word drawn by Authors

Therefore, the MSE essentially measures the quality of an underlying estimator. This is supported by the fact that it will always be positive, with values that arrive closer to zero being the better ones.

The MSE can be said to be the other step from the originating point of the error in question, and therefore, it will incorporate the following:

i. The width of the spread of the estimates from one point to the next, i.e., variance.

ii. The distance between the mean estimated values and the real observed values, i.e., the bias involved.

If the underlying estimator has zero bias, the mean squared error will act as the variance value of said estimation. MSE is, in a way, like a variance. This is due to the fact that it has similar estimation units as the estimation quantity, squared, which is also knows as the standard error.

Likewise, the Root Mean Squared Error (RMSE) or Root Mean Squared Deviation is a measure that is often used. It is used to measure the deviation between values or estimates calculated by a model and the actual values observed.

The Root Mean Squared Error is a representation of the square root of a second inline movement of a given sample. For these, it will be the differences between the values or estimates calculated by a said model and actual observations. It can also be a quadratic mean of these deviations. The deviations that are observed are known as losses, residuals, or errors. In a prediction problem, these will be called prediction errors in a computation scenario dealing with out-of-sample prediction cases.

The Root Mean Squared Error works by summing up the extent of the observed prediction errors over several time frames into one measure of tremendous prediction power. The Root Mean Squared Error, like the Mean Squared Error, aims to be a measure of accuracy and helps when a comparison of forecasting deviations amongst various models is required. The point to be noted is that although there will be different models involved, the measure will be for a particular dataset so as to allow for comparisons. This is due to the fact that this measure depends on the scale of the data involved (Brown and Spiegelman, 1991).

The Root Mean Squared Error will always produce a non-negative value. The measure can result in a zero value, but only in theory. This is because a zero value means a perfect fit for the data in question. As a general measure, the lower the recorded value of the data, the better it is said to be since it means in a prediction problem, the predicted values are closer to real-world values.

The Root Mean Squared Error is very susceptible to observed outliers. This is due to the fact that it is the square root of mean squared errors (MSE). Hence, the effect that each observed error will have on the Root Mean Squared Error will be proportional to the magnitude of their squared forms. Consequently, if the detected errors are significant, they will have an effect on the Root Mean Squared Error or Deviation that is inordinately large (Huang, 1999).

It should be noted regarding the error metrics that Mean Squared Error, Root Mean Squared Error, and Mean Absolute Error are measures of the aberration between the actual and predicted values. Hence, a forecast performance would be considered to be better when these values are lower.

The Mean Absolute Error (MAE) is simply put, a deviation between two continuous variables. This measure is prevalent in the analysis of time series and prediction problems (Willmott and Matsuura, 2005).

The table below shows all three metrices for both data sets as compared to a tuned ARIMA (Grillenzoni, 1993) model on the same.

Table 1 Metrices for data sets

	LENOVO			HONDA		
Metric	**MSE**	**RMSE**	**MAE**	**MSE**	**RMSE**	**MAE**
ARIMA	0.031	0.177	0.130	7034.5	83.871	63.13
LSTM	0.025	0.161	0.125	4037.6	63.543	45.341

Experiment results & discussion:

We split both data sets as 75% for training and 25% for testing. The data was converted to a binary feature range and scaled using sci-kit learn's min-max scaler. The scaled data was then fed to the neural network that consisted of a sequential setup containing two LSTM layers of 50 units each and a final Dense layer composed of one unit. The outputs were then used to calculate the metrices in the previous section and also for the prediction graphs shown below.

The prediction runs indicate that the model is indeed successful at predicting these datasets. The metrics, however, are very far apart due to currency differences. The graphs show that the model can predict the trends and movements of both stocks. Due to the nature of how LSTMs operate, however, they may be more successful in the long run. Different datasets, however, may make the model's performance worse or better.

Conclusion:

Any prediction activity has its downsides. All models need to have an underlying assumption, and the assumption in this model was that all previous data would be relevant. This, may not always occur, however.

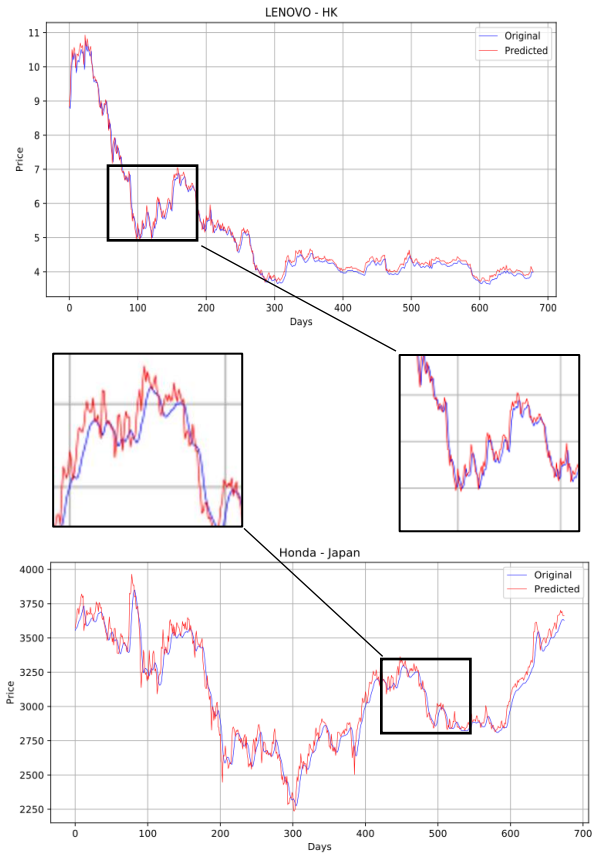


Fig 9 & 10: $\hat{Y}_i$ vs Y_i for Lenovo and Honda[10]

Therefore, the model may be tuned according to the problem at hand. Nevertheless, for datasets that have the same characteristics as the ones used in this study, the model should be very competitive.

Future additions:

Future work on the LSTM application can take several directions. Firstly, instead of daily prices, we could use up to the minute prices for improvement. Second, for the sake of simplicity, we have not applied any denoising to the datasets, while this could be done to improve overall data. One technique that can be very robust in this regard is wavelet denoising (Hazra and Guhathakurta, 2016).

Wavelet denoising or analysis has resulted in marvelous achievements in areas such as signal processing and image recognition and detection (Leduc, 1997). This technique is able to compensate for what is lacking in the Fourier analysis. Therefore, its use has slowly but surely made its way into the finance and economics fields.

[10]Source: Authors' Python outputs

This technique is being suggested since it has some outstanding advantages over traditional methods and techniques in the solving of time series analysis problems. Wavelet analysis can adequately carry out the decomposition and reconstruction functions on financial time series, even if they relate to different time frames or domain scales. Hence, the synergy attained by combining the wavelet analysis theories with traditional methods of time series analysis can allow for much better study and resolution of problems in financial data.

Furthermore, the data input could be made perpetual by making use of time windows that act as inputs and move the training forward in time. This could have a positive impact on both long and short term predictions.

Finally, a hybrid model, along with technical indicators, would go a long way into reducing the distance from actual values.

These are being suggested since hybrid models and the synergies they bring make for some very robust and powerful tools for Management Science (MS) tools practitioners. The literature has evidence of this. Hong et al., (2015) suggested a hybrid forecasting model to predict parameters for local traffic flow. Moreover, hybrid models were used by Ribeiro and Oliveira (2011) for predicting the prices of agricultural products.

Therefore, we can safely conclude that on a broader scale, it has been proven that there does indeed exist a synergy in the use of hybrid models whereby forecasting quality can be hugely improved. This is because a combination of models can be more robust in capturing the highly complex relationships among dependent and independent variables.

Author Contributions:

Methodology: Khalid Saboor & Han Liyan

Writing (Initial): Khalid Saboor

Methodology Review: Qurat Ul Ain Saboor

Review & Editing: Abdul Saboor Zahid

Disclaimer:

The above contributions indicate significant parts of the work involved. All authors were engaged, directly or indirectly, in all parts of this study.

Appendix:

This section lists out some python code extracts for reference purposes. All code is either self-written or sourced from open-source repositories (GitHub, 2020).

Libraries:

```
#importing required packages
#to suppress all warnings
import warnings
warnings.filterwarnings('ignore')
#standard imports
import pandas as pd
```

```
import numpy as np
import urllib.request as urb
import collections
from math import sqrt
#to do plotting within jupyter
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
#we set the figure size here (for all figures that we plot)
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 20,10
#MinMaxScalar for data normalization
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))
#required by LSTM
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, Dropout, LSTM
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import accuracy_score
```

Data:

```
#reading data file into a variable
data=pd.read_csv(r'B:\Other\share\LVOG.csv')
data.drop('Unnamed: 0', axis=1, inplace=True)
data.head(5)
```

Length Check:

```
#Data Length : Lenovo
print(len(data))
Tlen= len(data)*.75
print(Tlen)
print(len(data)-Tlen)
```

Split:

```
# L S T M
warnings.filterwarnings('ignore')
#dataframe
data = data.sort_index(ascending=True, axis=0)
new_data = pd.DataFrame(index=range(0,len(data)),columns=['Date', 'Adj Close'])
for i in range(0,len(data)):
    new_data['Date'][i] = data['Date'][i]
    new_data['Adj Close'][i] = data['Adj Close'][i]
```

```
#setting index
new_data.index = new_data.Date
new_data.drop('Date', axis=1, inplace=True)
#splitting the dataset into two datasets for trianing and testing
#Lenovo
lstm = new_data.values
train = lstm[0:2037,:]
valid = lstm[2037:,:]
#converting dataset into x_train and y_train
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(lstm)
x_train, y_train = [], []
for i in range(679,len(train)):
    x_train.append(scaled_data[i-679:i,0])
    y_train.append(scaled_data[i,0])
x_train, y_train = np.array(x_train), np.array(y_train)
x_train = np.reshape(x_train, (x_train.shape[0],x_train.shape[1],1))
print(len(valid)) #to check how many values we will be predicting
```

LSTM Network:

```
# initializing and fitting the LSTM network
warnings.filterwarnings('ignore')
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(x_train.shape[1],1)))
model.add(LSTM(units=50))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='nadam')
history=model.fit(x_train, y_train, epochs=3, batch_size=1, verbose=1)
#predicting 679 values, using past 2037 from the train data
inputs = new_data[len(new_data) - len(valid) - 679:].values
inputs = inputs.reshape(-1,1)
inputs  = scaler.transform(inputs)
X_test = []
for i in range(679,inputs.shape[0]):
    X_test.append(inputs[i-679:i,0])
X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0],X_test.shape[1],1))
closing_price = model.predict(X_test)
closing_price = scaler.inverse_transform(closing_price)
print ("All Epochs Succeeded")
```

Errors & Plots:

```
#Errors
rmse=np.sqrt(np.mean(np.power((valid-closing_price),2)))
rmse
def mape(y_pred,y_true):
```

```
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
mse = mean_squared_error(valid,closing_price)
rmse = sqrt(mean_squared_error(valid,closing_price))
mae=mean_absolute_error(valid,closing_price)
maperror=mape(valid,closing_price)
print('MSE')
print(mse)
print('RMSE')
print(rmse)
print('MAE')
print(mae)
print('MAPE')
print(maperror)
#plots
plt.figure(figsize=(10,5))
plt.grid()
plt.title('LENOVO - HK')
plt.plot(closing_price,color='blue',label='Original',linewidth=0.5)
plt.plot(valid, color='red',label='Predicted',linewidth=0.5)
plt.xlabel("Days")
plt.ylabel("Price")
plt.legend()
plt.savefig(r'B:\Other\share\lstmfigs\LVOG.svg')#str(data)+'_'+str(ratio)+'.svg')
plt.show()
```

References:

Brown, P., and Spiegelman, C. (1991). Mean squared error and selection in multivariate calibration. *Statistics & Probability Letters*, 12(2):157-159.

Cai, M., and Liu, J., 2016. Maxout neurons for deep convolutional and LSTM neural networks in speech recognition. *Speech Communication*, 77, pp.53-64.

Chauhan, B., Bidave, U., Gangathade, A., Kale, S. (2014). Stock Market Prediction Using Artificial Neural Networks. *International Journal of Computer Science and Information Technologies*, 5(1): 904-907

Dingli, A., and Fournier, K. (2017). Financial Time Series Forecasting – A Deep Learning Approach. *International Journal of Machine Learning and Computing*, 7(5):118-122.

El-Keib, A., Ma, X. and Ma, H. (1995). Advancement of statistical-based modeling techniques for short-term load forecasting.*Electric Power Systems Research*, 35(1):51-58.

Erdinc, A. (2005). Stock Market Forecasting: Artificial Neural Network & Linear Regression Comparison in Emerging Market. *Journal of Financial Management & Analysis*, 18(2):18-33.

Girirajan, S., Sangeetha, R. and Preethi, T., and Chinnappa, A. (2020). Automatic Speech Recognition with Stuttering Speech Removal using Long Short-Term Memory (LSTM). *International Journal of Recent Technology and Engineering*, 8(5):1677-1681.

GitHub. (2020). *Build Software Better, Together*. Retrieved February, 8, 2020 from https://github.com/

Grillenzoni, C. (1993). ARIMA Processes with ARIMA Parameters. *Journal of Business & Economic Statistics*, 11(2): 235.

Hazra, T. K., and Guhathakurta, R. (2016). Comparing Wavelet and Wavelet Packet Image Denoising Using Thresholding Techniques. *International Journal of Science and Research*, 5(6): 790-796.

Hochreiter, S., and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8):1735-1780.

Hong, H., Huang, W., Xing, X., Zhou, X., Lu, H., Bian, K., and Xie, K. (2015). Hybrid Multi-metric K-Nearest Neighbor Regression for Traffic Flow Prediction. Retrieved July 24, 2020 from https://ieeexplore.ieee.org/document/7313457/authors#authors

Huang, J. (1999). Third-order expansion of mean squared error of medians. *Statistics & Probability Letters*, 42(2):185-192.

Kim, T., and Kim, H., 2019. Forecasting stock prices with a feature fusion LSTM-CNN model using different representations of the same data. *PLOS ONE*, 14(2), p.e0212320.

LeCun, Y., Bengio, Y. and Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436-444.

Leduc, J. (1997). Spatio-temporal wavelet transforms for digital signal analysis. *Signal Processing*, 60(1):23-41.

Lee, D., and Priebe, C. (2000). Exact mean and mean squared error of the smoothed bootstrap mean integrated squared error estimator. *Computational Statistics*, 15(2):169-181.

Maia, C., and Gonçalves, M. (2008). Application of switched adaptive system to load forecasting. *Electric Power Systems Research*, 78(4):721-727.

Meharia, V. (2011). Financial Time Series Forecasting Through Support Vector Machines. *SSRN Electronic Journal.*

Pavlyuchenko, B., (2019). Machine-Learning Models for Sales Time Series Forecasting. *Data*, 4(1):15.

Reddy, B., and J. C, U., (2019). Prediction of Stock Market using Stochastic Neural Networks. *International Journal of Innovative Research in Computer Science & Technology*, 7(5):128-138.

Ribeiro, C. and Oliveira, S. (2011). A hybrid commodity price-forecasting model applied to the sugar-alcohol sector. *Australian Journal of Agricultural and Resource Economics*, 55(2):180-198.

Sharaf, A. and Lie, T. (1995). A novel neuro-fuzzy based self-correcting online electric load forecasting model. *Electric Power Systems Research*, 34(2):121-125.

Tang, B., Wang, X., Yan, J., and Chen, Q. (2019). Entity recognition in Chinese clinical text using attention-based CNN-LSTM-CRF. *BMC Medical Informatics and Decision Making,* 19 (S3).

Uchida, S., Miyazaki, H., and Sakoe, H. (2008). Mosaicing-by-recognition for video-based text recognition. *Pattern Recognition*, 41(4), 1230-1240.

Wang, J., Wang, J., Zhang, Z., and Guo, S., 2012. Stock index forecasting based on a hybrid model. *Omega*, 40(6), pp.758-766.

Wang, Xi-Ping & Meng, Ming. (2008). Forecasting electricity demand using the Grey-Markov model. Proceedings of the 7th International Conference on Machine Learning and Cybernetics, ICMLC. 3. 1244 - 1248. 10.1109/ICMLC.2008.4620595.

Willmott, C., and Matsuura, K. (2005). Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance. *Climate Research*, 30:79-82.

Xiao, C., Xia, W., and Jiang, J., (2020). Stock price forecast based on combined model of ARI-MA-LS-SVM. *Neural Computing and Applications*. 32, 5379–5388

Zhang, B., and Dong, Z. (2001). An adaptive neural-wavelet model for short term load forecasting. *Electric Power Systems Research*, 59(2), 121-129.