A Comparative Study of LSTM and DNN for Stock Market Forecasting

Dev Shah
School of Computing
Queen's University
Kingston, ON, Canada K7L 2N8
16ds57@queensu.ca

Wesley Campbell
Department of Civil Engineering
Queen's University
Kingston, ON, Canada K7L 2N8
w.campbell@queensu.ca

Farhana H. Zulkernine
School of Computing
Queen's University
Kingston, ON, Canada K7L 2N8
farhana@cs.queensu.ca

***Abstract*—Prediction of stock markets is a challenging problem because of the number of potential variables as well as unpredictable noise that may contribute to the resultant prices. However, the ability to analyze stock market trends could be invaluable to investors and researchers, and thus has been of continued interest. Numerous statistical and machine learning techniques have been explored for stock analysis and prediction. We present a comparative study of two very promising artificial neural network models namely a Long Short-Term Memory (LSTM) recurrent neural network (RNN) and a deep neural network (DNN) in forecasting the daily and weekly movements of the Indian BSE Sensex index. With both networks, measures were taken to reduce overfitting. Daily predictions of the Tech Mahindra (NSE: TECHM) stock price were made to test the generalizability of the models. Both networks performed well at making daily predictions, and both generalized well to make daily predictions of the Tech Mahindra data. The LSTM RNN outperformed the DNN in terms of weekly predictions and thus, holds more promise for making longer term predictions.**

Keywords; Artificial neural networks, deep learning, long short-term memory, multi-layer neural network, recurrent neural network, financial forecasting, stock market analysis

I. INTRODUCTION

Following the well-known "Efficient Market Hypothesis" (EMH) proposed by Fama [4], it has long been stated that the prediction of stock markets is an impossible task. The number of potential variables influencing the resultant stock or index value, as well as the fact that human sentiments drive the behavior of markets explains why stock prediction is not an easy task. Despite this, investors have been attempting to predict the future values, or at least the future trends of stock markets. Over the years, some investors like Warren Buffet have beaten the markets consistently, thereby challenging the EMH. It is certainly possible to find opportunities for making gains because human sentiments and certain events (news, government policies, etc.) may temporarily move prices away from the market equilibrium. Over-reactions may occur causing excessive optimism to drive prices very high, or excessive pessimism to drive prices unduly low. Such inefficiencies give rise to the adoption of a wide array of approaches for making profits by identifying stock market trends even if these methods do not identify the exact prices.

Initially, statistical models like the Auto Regressive Integrated Moving Average (ARIMA) and linear regression were employed for predicting stock prices. Ariyo et al. [1] explored the extensive process of building ARIMA models. The best ARIMA model did a satisfactory job at predicting the stock prices of Nokia and Zenith Bank. However, since the rise of machine learning, researchers have focused on, and explored various techniques such as the artificial neural networks and genetic algorithms to predict time series data. As a proof of concept, Shen et al. [10] proposed a prediction algorithm which exploited correlations among global markets and other products to predict the next day trend of stock prices using a machine learning approach called Support Vector Machine (SVM). Shen et al. chose varied datasets which might have impact on each other like USD, JPY, NASDAQ, Gold, Oil, etc. and identified important features via measures such as auto- and cross-correlation. Results of the study boasted a 77.6% prediction accuracy on the Dow Jones Industrial Average (DJIA) and up to 85% for longer term predictions of greater than 30 days. Wang et al. [13] have also demonstrated that a combination of statistical and machine learning techniques also can work well to predict time series data.

Neural networks are models which connect numerous artificial neurons (each performing a simple computational task) together to create a complex model which can learn to recognize trends in the data, extract features, or to represent them in a compressed form. Deep neural networks (DNN) are neural networks with more than one hidden layer of neurons. The embedded layers in a DNN allow it to represent data with a very high level of abstraction, depending on the number of layers in the network. However, attempting to predict time series data such as stock markets is still a challenging task, and DNNs may still be limited in their ability to predict such data. When working with sequential data, one cannot process related time series at every time step and save the entire state of the sequence. This is a task for which recurrent neural networks (RNN) are better suited. RNNs pass input data to the network across time steps, hence processing one element at a time. Recurrent networks allow for information to persist, hence solving the problem of "forgetting" previous inputs. Basic RNNs perform particularly well in modeling short sequences. They have hidden layers which take the current state's input and the previous state's hidden layer output as inputs. Nonetheless, they have a problem of vanishing

978-1-5386-5035-6/18/$31.00 ©2018 IEEE

gradient (memory of older data becomes subtle, causing the gradient signal to become very small), where learning can become very slow for long-term dependencies in the data. On the other hand, if the values in the weight matrix become large, this can lead to a situation where the gradient signal is so large that the learning scheme diverges. This phenomenon is often called as "exploding gradient."

To overcome the problem of vanishing gradient, an interesting approach called Long Short Term Memory (LSTM) model was developed by Hochreiter and Schmidhuber [6]. Compared to simpler RNNs, LSTM replaces neurons with a memory cell. LSTM's single time step cell has a more complex structure consisting of a hidden state, an input and an output [6]. Inside these cells known as memory blocks, there are three adaptive and multiplicative gating units: the input gate, the forget gate and the output gate. Both input and output gates have the same role as in a simple RNN. The forget gate learns how to remember and what to remember, based on the input data. The internal state feeds into itself across time steps with a constant weight of 1. This eliminates the vanishing gradient problem since any error that flows into the self-recurring unit during backpropagation is preserved indefinitely. Each gate implements an activation function (such as a sigmoid function). In the forward pass, input gates learn when to let activation pass into the cell and output gate learns when to let activation pass out of it to the rest of the network. In the backward pass the output gates learn when to let error flow into the cell and the input gate learns when to let error flow out of the cell to the rest of the network. This allows the network to remember long term dependencies.

This work serves to compare two popular artificial neural network models namely the DNN and the LSTM RNN, in their abilities to make daily and weekly predictions of the value of the Bombay Stock Exchange Index (BSE Sensex), as well as daily predictions of the Tech Mahindra stock price. The rest of the paper is organized as follows. Section II presents a brief survey on the usage of DNN and RNN. Section III describes the data and Section IV describes the models we implemented. Results and a critical discussion are presented in Sections V and VI respectively. Section VII draws the conclusion.

II. Related Work

Here, we will go through some of the most recent works which use DNN and RNN variants for predicting time series data.

A. Deep Neural Networks

Several studies have investigated variations of DNNs to predict time series data. One such variation observed in the literature is the usage of unsupervised learning to pre-train a network, followed by a common supervised technique to optimize the weights and biases. Some studies have utilized a combination of Restricted Boltzmann Machines (RBM) and a Multi-Layer Perceptron (MLP) [7][8]. In these studies, RBMs were trained in an unsupervised manner in a "feature extraction phase" with the weights of the MLP being trained afterwards using backpropagation to optimize the predictive capacity of the network. Kuremoto et al. [8] used networks with one and two RBMs respectively, followed by a single MLP to predict Lorenz chaos and a Henon map. Both networks outperformed a typical MLP network, and a deep belief network (i.e., only RBMs) on its own. Kuremoto et al. [7] used two RBMs and a MLP to predict time series data. The structure of each network in this study was optimized using Particle Swarm Optimization (PSO) algorithm. The combined RBM-MLP networks outperformed the simple MLP model.

Singh and Srivastava [11] used a DNN to predict stock prices, whose inputs were selected using Principal Component Analysis (PCA). They used opening, closing, high, and low prices of stocks, as well as 36 other input variables. The DNN had two hidden layers, and the weights were trained using backpropagation. The adaptive learning rate algorithm (ADADELTA) [12], which adds momentum and learning rate annealing to a stochastic gradient descent algorithm, was used to optimize the training. Singh and Srivastava [11] compared their network to a Radial Basis Function (RBF) network and a RNN. They showed that their network outperformed both the RBF and the RNN.

B. Recurrent Neural Networks

Recently, there has been research done on the application of RNNs and the LSTM variant on stock price analysis and prediction. Di Persio and Honchar [3] implemented and compared three different variants of RNNs for forecasting Google's stock price. The study showed that LSTM did a better job when compared to the basic RNN and Gated Recurrent Unit (GRU) architectures and achieved an accuracy of 72% for a 5-day period. Di Persio and Honchar [3] made sure that the network was not overfitting by shuffling training and testing sets after splitting. Thus, results appear to be robust. The authors also presented insights into how RNN works for finding patterns by providing a visualization of the internal working of the RNNs at run time.

Roondiwala et al. [9] implemented a LSTM RNN to predict the movements in the Nifty Index, leveraging features like open, close, high and low prices. This approach resulted in a root mean squared error (RMSE) of 0.0086 after training for 500 epochs. However, the paper failed to mention how they avoided overfitting and did not display the actual results. In the study, the authors presented the RMSE in terms of daily percent changes in the stock price. The value is near zero but the absolute value of RMSE would be around 1% of the value of Nifty index which is slightly greater than 100. Bernal et al. [2] used Echo State Networks (ESN), a subclass of RNN, to predict stocks in the S&P 500. The authors used descriptive features like price, moving averages and volume as input attributes and achieved a test error of 0.0027 (daily price change). Compared to the Kalman filter, its performance was better and was able to better predict rapid changes (volatility) in the index.

There are a few studies addressing the question of why LSTM networks perform well, how they compare against deep learning networks, or why and when should a LSTM RNN be preferred over a DNN. This works aims to do just this, by comparing the performance of a LSTM RNN with that of a DNN when applied to predicting stock prices.

III. DATA PREPROCESSING AND NORMALIZATION

Before We used a web[1] data set containing twenty years of BSE Sensex daily close prices from 1997-2017 in this study. In order to facilitate an even comparison between the LSTM and DNN networks, we used only the daily closing value of this stock index for prediction. For each network, the data was initially split into input and output data sets based on a specified window size. For example, daily predictions with a window size of 20 would mean that the data point at time *t* was predicted using the previous 20 data points (*t-1, t-2, … , t-19, t-20*). Different window sizes were used for each network. Initially, a window size of 20 was chosen for the DNN. A window size of 50 was chosen for the LSTM RNN. The window sizes were chosen based on each model's suitability with the price data. Both models were tried with different window sizes, but the final values did not have any impact in terms of one model outperforming the other.

After the input and output sets were constructed, we normalized the data using Eq. 1. Data values were converted to "relative change" over the specified window, relative to the first value in each window as shown in Eq. 1. Predicting percent change instead of actual value is more suitable and therefore normalizing the data across windows allows for the optimization process to converge. Since different windows were used for each network, the values of relative change differed slightly for each network at this point.

$$n_i = (a_i / a_o) - 1 \quad (1)$$

where:

$n_i = i^{th}$ normalized value

$a_i = i^{th}$ closing value

a_o = initial closing value in window

For the LSTM RNN, the values of relative change were not altered after this point. However, one more step was carried out for the DNN. Since the DNN employed Rectified Linear Unit (ReLU) activation functions (Eq. 2) in its hidden layers, it could not handle values less than zero. To evade this problem, the values of relative change were converted to a [1,0] scale within each window based on the maximum and minimum values within the window. The normalized data was split into 80% training, 10% validation and 10% testing data sets. For the LSTM RNN, the training data was shuffled before training the network. This non-sequential training method was applied to improve the generalizability of the network. All preprocessing and normalization was done using Python.

Fig. 1 and 2 demonstrate the data flow of the models. Fig. 1 gives a brief overview of the system. On the other hand, Fig 2 explains the system in detail by breaking down the '1.0' process of NN model into further sub processes viz. Preprocessor, Normalizer, Partition and Build Model. Preprocessor and normalizer processes are explained in Section III and partitioning refers to the splitting of data into training, validation and testing data sets. The implementation details of both the models will be discussed in the next section.

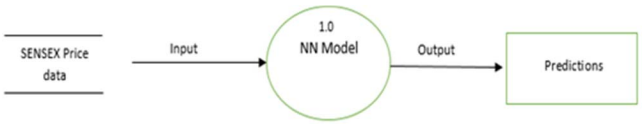

Figure 1. Level 0 Data flow diagram.

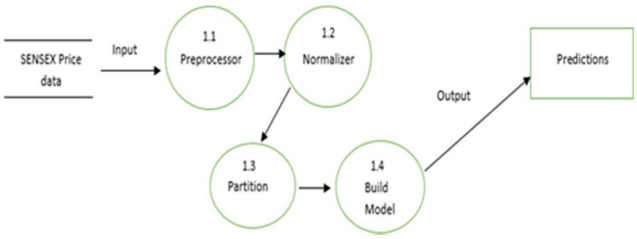

Figure 2. Level 1 Data flow diagram

IV. IMPLEMENTATION

A. Design Approach

1) Deep Neural Network

To obtain an idea about how quickly this network learned and minimized its training error, the first and second hidden layers were initially designed with 40 and 20 hidden neurons respectively. The network was trained for an arbitrary 100 epochs and tested. This initial test run was done using a simplified data split of 90% training and 10% testing, a window of 20 input values, and data that was normalized to a [1,0] scale without first expressing in terms of relative change over each window. Dropout was not applied for this exploratory run. Upon observation of the training error during this run, the point at which the MSE was sufficiently minimized was chosen to be the 30-epoch point. This length of training interval was selected as the interval over which to compare subsequent versions of the network with different numbers of hidden nodes in each layer.

Since the decision of constructing a 20-40-20-1 network architecture was arbitrary, it was reasonable to presume that there could be a network architecture that could predict stock prices with the same level of accuracy or better, but in less time and using fewer computational resources. Using the constant training period of 30 epochs, each possible network having an architecture 20-X-Y-1 (where X and Y are both integers on the interval [1,40]) was trained and tested. Thus, a total of 1,600 network architectures were trained and tested. Again, dropout was not applied to these runs. For each network, the prediction results on the testing data and the MSE during training were plotted. Furthermore, the forecast bias of the prediction relative to the testing data was plotted against the number of neurons in the second hidden layer, for each number of neurons in the first hidden layer. Each of the 40 plots of forecast bias vs. the number of second hidden layer nodes was examined, and network architectures yielding in a forecast bias between -10 and 10 (absolute BSE Sensex value) were selected. The number of hidden neurons in each of these

[1] https://www.bseindia.com

networks was recorded, and each one of these networks was trained and tested again, this time recording the time of training in seconds. The training time was then plotted against the total number of nodes in each network with a color bar indicating the forecast bias of each network as shown in Fig. 3.

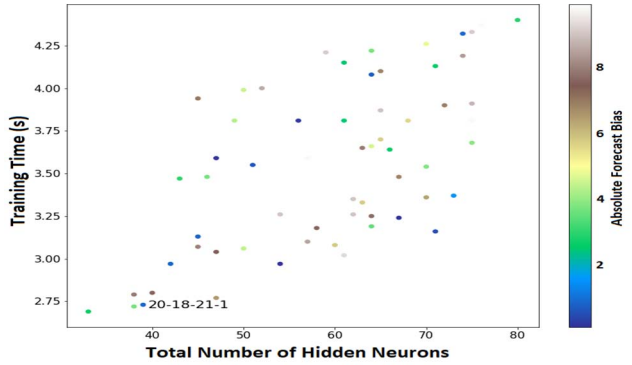

Figure 3. Determining the most efficient DNN

From these results, the network configuration 20-18-21-1 (20 inputs, 18 hidden nodes in the first layer, 21 hidden nodes in the second layer, and 1 output) was selected based on the fact that it was nearly the most efficient in terms of computation time and total nodes, as well as having yielded a forecast bias of nearly zero when trained for 30 epochs. Using this configuration of 18 and 21 hidden nodes, the network was then refined. To optimize the training of this network, the fully preprocessed and normalized data (as described in Section III) was used with dropout being added to the training regime.

This network was trained and tested for a variety of window sizes, each for 30 epochs. Based on the plots of MSE during training and validating, a final network of 19-18-21-1 was selected for daily predictions of the BSE Sensex data. Refining the number of inputs to 19 allowed the training and validation errors to coincide and stabilize as shown in Fig. 5a. The training period was then refined to 18 epochs to avoid overtraining.

Using the same network, weekly (7 trading days ahead) predictions were attempted. Weekly predictions were made using the DNN by making a series of daily predictions. The difference here was that once the first day of a given week was predicted, that predicted value was then included in the inputs for the second day of that week. Thus, the seventh day of each weekly prediction series was predicted using 13 true values and 6 predicted values. The network, therefore, made predictions using its previous predictions. Further, using the same number of hidden layers (18 and 21), this network was tested in its ability to make daily predictions of the Tech Mahindra stock price. Using only the closing price (as done with the BSE Sensex), the DNN was trained, tested and refined in the same manner. It was found that the training and validation errors stabilized using an input window of 8, and 25 epochs of training as shown in Fig. 8a. Thus, a network with the configuration 8-18-21-1 was used to make daily predictions of the Tech Mahindra stock price, testing the generalizability of the DNN.

2) Long Short-Term Memory Network

The LSTM neural network was trained and tested with different configurations of parameters to optimize the fit based on the forecast bias and Root Mean Squared Error (RMSE). Initially, the network was trained and tested using 50 and 100 LSTM nodes respectively on a 10-year dataset of the BSE Sensex. However, after training and testing it was observed that increasing the training dataset to the full 20 years provided the network a larger and more varied dataset to learn from, but this also made it more difficult for the network to learn. For this reason, the number of nodes in the second layer was increased to 200 to more accurately learn from the training data.

The model was trained for a range of 15-35 epochs, and training was done batch wise where each batch size was 200. Generally, a lower batch size is better for a network like a LSTM RNN, both in terms of learning accuracy as well as in terms of computational resources. An early callback function was implemented which dynamically stopped the training as soon the validation loss stopped decreasing. A "patience level" for the callback function was set to 3 which allowed the network to initially accept increases in validation loss (an expected result when using the dropout technique). Using the same network, daily predictions were made on the Tech Mahindra data with a batch size of 50. For weekly predictions, RMSPROP [15]., a variant of backpropagation, was employed. RMSPROP is an unpublished but widely used adaptive learning rate method. Adjusting the learning rate from the default value of 0.001 to 0.0005 and weight decay rate from 0.0 to 0.05 allowed the network to nicely converge for weekly predictions.

B. Network Configuration

1) Deep Neural Network

The DNN was constructed using the Keras package in Python at the front-end and Tensorflow at the back-end. A fully connected network consisting of 4 layers, one input, two hidden, and one output layer, was constructed. The first hidden layer initially had 40 neurons, and the second hidden layer had 20 neurons. Based on the findings of Glorot et al. [5], the ReLU function as shown in Eq. 2 was used for the activation of the hidden neurons. The sigmoid function was used to activate the output node, ensuring that the output was between values of 0 and 1. Unlike previous studies, this network was not pre-trained using unsupervised learning or statistical techniques. However, this is justified by Glorot et al. who found very little difference in neural network performance in image recognition when a ReLU-activated network was pre-trained with unsupervised learning compared to the untrained version. Not having to pre-train the network also potentially saves computational time.

$$f(net) = max(0, net) \tag{2}$$

The DNN employed the adaptive learning rate algorithm (ADADELTA) by Zeiler [12], which combines momentum and learning rate annealing with stochastic gradient descent to train the weights. Weights were initialized using a random

normal distribution with a mean value of 0 and standard deviation of 0.05. To ensure reproducibility across model runs, the seed of the random number generator was arbitrarily fixed to a value of 4. Mean squared error (MSE) was used as the loss function during training and validation.

A dropout strategy was applied to avoid overfitting. Dropout is an explicit regularization technique which drops certain nodes in the network in an attempt to introduce noise in the data such that the network does not overfit to the data. Dropout can be applied at the input layer as well as at the hidden layers. In this case, after experimentation a specified 50% of nodes were purposely dropped during training in both the hidden layers.

2) Long Short-Term Memory Network

The LSTM model was constructed using the same Keras package with Tensorflow as back-end. The network configuration was chosen through thorough experimentation. The network consisted of 4 layers, one input, two hidden, and one output layer. As described in Section 3, this network was given 50 input values for every output. The first hidden layer was initially given 50 LSTM cells, and the second hidden layer 100. A linear activation function was used, and the model was compiled with the ADAM optimizer [14] which is a first order gradient based optimization technique because it was observed that ADAM optimizer was able to minimize the validation loss and RMSE to a greater extent when compared to RMSPROP [15]. MSE was used as the loss function. An early stopping technique was used to stop the network training once the error ceased to drop on validation samples. Initially a 50% dropout was implemented to avoid overfitting and improve generalizability. However, after testing a 25% dropout was used as it ensured appropriate training of the network and avoided premature convergence.

V. RESULTS

Figures 4 to 8 display the results of daily and weekly (7-day ahead) BSE Sensex predictions, as well as daily Tech Mahindra predictions. To compare the predictive abilities of both models, the daily predictions of the DNN and the LSTM RNN were compared using two metrics: RMSE and forecast bias. RMSE is considered to be a good metric for analyzing time series predictions and an ideal RMSE would be zero. Forecast bias gives an indication of the model's directional bias (higher or lower) with respect to the true values. A model with a large positive forecast bias would therefore be expected to consistently overpredict the true data. Thus, a fair, unbiased model would yield a forecast bias of zero. Directional accuracy (DA) was used to gauge the performance of weekly predictions. DA is similar to a binary evaluation because it compares the direction of each prediction against the direction of true data for a specified time period i.e. 7 days and takes an average across all predictions. Tables I to III compare RMSE and forecast biases for both sets of daily predictions, and directional accuracy (DA) for the weekly predictions. The values of the metrics like RMSE, bias and DA represent the average values of fifty iterations. In Fig. 6a and 6b, only 20-30 weeks of predictions are shown for clarity.

TABLE I. COMPARISON OF RMSE AND FORECAST BIAS, CALCULATED FROM THE DENORMALIZED DAILY PREDICTIONS OF THE RAW BSE SENSEX DATA.

Network	RMS Error (raw BSE Sensex value)	Forecast Bias (raw BSE Sensex value)
DNN	200	65
LSTM RNN	238	-50

TABLE II. COMPARISON OF THE DIRECTIONAL ACCURACY FOR WEEKLY PREDICTIONS, CALCULATED FROM THE NORMALIZED (RELATIVE CHANGE) BSE SENSEX DATA.

Network	Directional Accuracy
DNN	0.526
LSTM RNN	0.606

TABLE III. COMPARISON OF RMSE AND FORECAST BIAS FOR NSE: TECHM

Network	RMS Error (raw Tech Mahindra value)	Forecast Bias (raw Tech Mahindra value)
DNN	6.33	1.09
LSTM RNN	9.72	-1.99

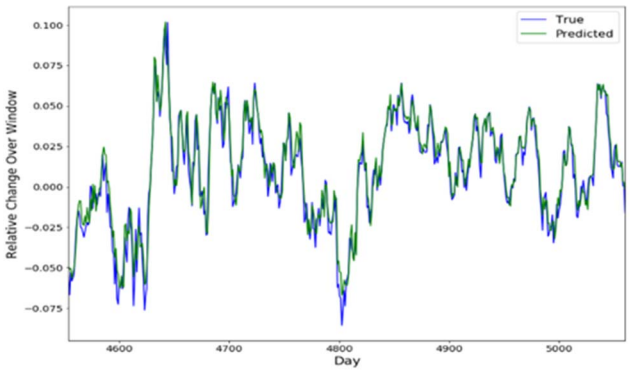

Figure 4a. Daily BSE Sensex predictions using the DNN

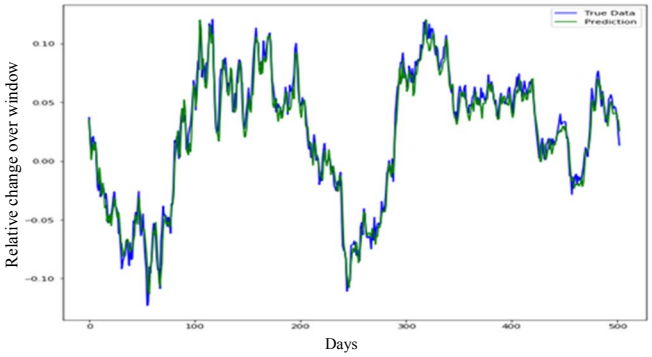

Figure 4b. Daily BSE Sensex predictions using the LSTM

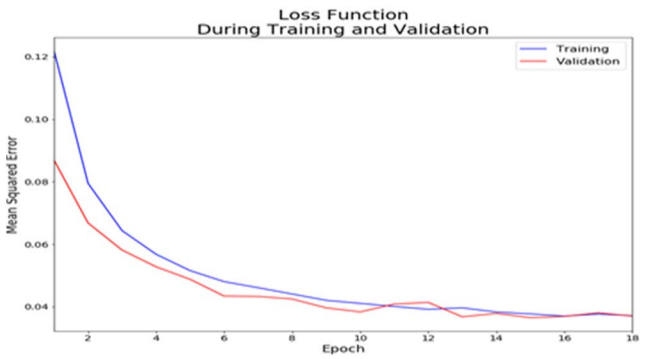

Figure 5a. Training and validation losses of DNN on BSE Sensex

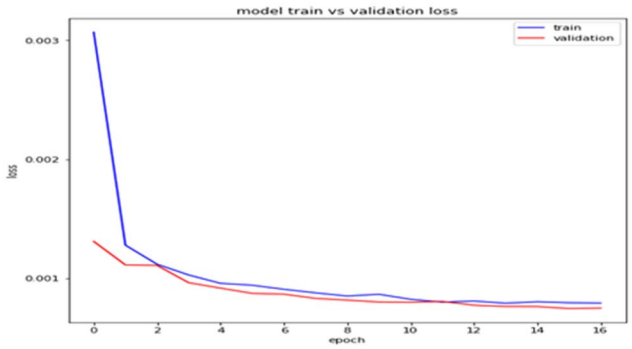

Figure 5b. Training and validation losses of LSTM on BSE Sensex

VI. Discussion

A. Daily Predictions

Fig. 4a and 4b show that the DNN and LSTM RNN both make daily predictions reasonably well, closely tracking the true data. Fig. 4a and 4b both express the daily predictions in relative change for the specified window size of each network. Each model was initially trained with both 80:10:10 and 70:10:20 (train:validate:test) data splits. The 80:10:10 split allowed the training and validation errors to coincide and stabilize more easily. Therefore, this split was chosen to compare the networks. If a model is well trained, its performance should be similar on both the training and validation sets. In both Fig. 5a and 5b, the training and validation errors stabilize at 18 and 16 epochs, respectively. The losses plotted are in terms of normalized values. These plots demonstrate the evidence that the networks were not overtrained, and thus provide a good fit to the data.

From the results in Table I it can be seen that the DNN produced a lower RMSE than the LSTM RNN, and the LSTM RNN resulted in a lower absolute forecast bias. However, both models appear to have produced robust results. The average value of the BSE Sensex over the testing data interval was around 28,000. The RMSE values in Table I are only about 1% of this average value. Considering that this dataset is noisy and non-uniform, the fact that both models managed to achieve forecast biases of this magnitude indicate that both performed well. This leads us to believe that daily predictions can be made with comparable accuracy using either the DNN or the LSTM RNN.

B. Weekly Predictions

The daily predictions look encouraging, however the reason that the DNN and LSTM RNN do so well is likely the fact that both models make point-by-point predictions. This means that the value at time t was predicted using the true values at time t-1, t-2, …, t-n, where n is the window size. The networks were therefore not predicting based on any previous predictions. The true test of their capabilities would be to attempt to predict multiple days (e.g., 1 week or 1 month) ahead.

Fig. 6a and 6b display how the DNN and LSTM models work for weekly (7 trading days ahead) predictions on BSE Sensex. The DNN does a decent job in making these predictions, however, it has difficulties recognizing the quick changes characteristic of this type of time series data. In the cases where the trend of the true data was very clear, the DNN did well at times to predict both the magnitude and direction of this trend. However, when there were multiple abrupt changes within a week's time of true data, the DNN struggled. This is in line with our expectations because the models were using only historical price for prediction. Also, the DNN is not capable of having a "memory" and hence it is expected that any rapid moves could not be predicted over a 7-day period without overfitting.

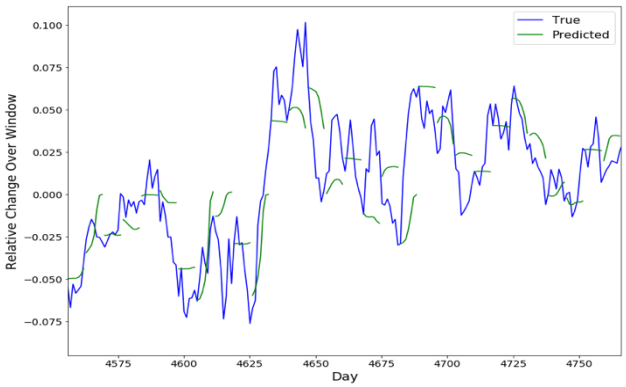

Figure 6a. DNN weekly predictions on BSE Sensex

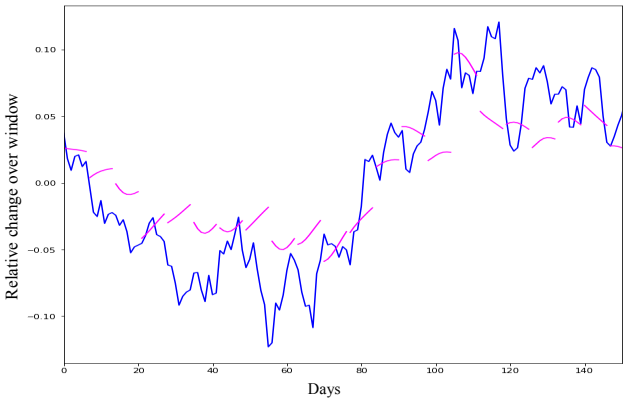

Figure 6b. LSTM weekly predictions on BSE Sensex

From Fig. 6b it can be seen that LSTM RNN does a better job at predicting the weekly movements. In more than half of the cases LSTM RNN was able to predict volatile movements in the true data. In general, the LSTM RNN was better able to recognize the directionality of the changes in the true data. This is supported by the fact that the LSTM RNN achieved higher directional accuracy. LSTM is less accurate for some weeks in the start because of higher volatility. Also, upon observation of Fig. 6b, a slight delay can be observed when compared to the actual data. This is due to the fact that the LSTM RNN was predicting on the predictions. Therefore, as it predicts more and more into the future, the error in the previous predictions amplify further. It can be seen that for some weeks LSTM performs better than Thus, LSTM RNN's are able to learn long term dependencies in the dataset. Therefore, as this type of model is trained on more and more data, it could be possible for this delay to diminish.

C. *Generalizability*

Measures were taken to prevent each of the networks from overfitting to the training data. Despite having learned the training data set more accurately, an overfitted or overtrained neural network would not perform as well on new data because of the fact that it learned the training data (including all of its outliers and atypical trends) too well. A simple measure taken to evade overfitting was to stop the training of each of the networks when the values of the loss function of the training data and validation data sets stabilized. This stabilization can be seen in Fig. 5a, 5b, 8a and 8b. A dropout strategy was also employed for both networks to avoid overfitting and improve each network's generalizability. Generally, stock indices are less volatile and hence considered safer for investment when compared to individual stocks. To test the generalizability of these models, they were tested on a new stock dataset (Tech Mahindra, NSE: TECHM) as shown in Fig. 7a and 7b. It was observed that both the models performed relatively well in making their predictions, considering the higher volatility of the Tech Mahindra stock. Therefore, the results from the daily Tech Mahindra predictions validate that the models here are not only suited to make daily predictions of the BSE Sensex, but that they also can generalize well for new data.

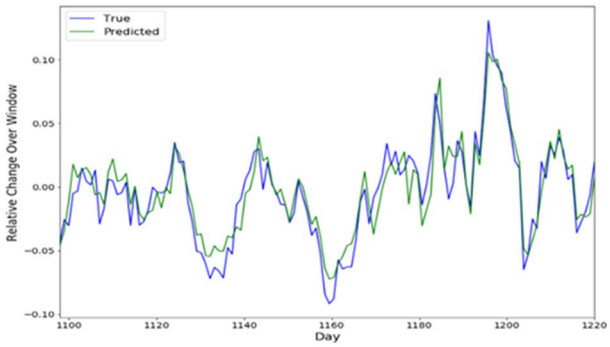

Figure 7a. DNN daily predictions on NSE: TECHM

The prediction results were plotted on a normalized scale (relative change over the input window) which allowed for easier comparison to other studies that used entirely different data sets. However, in this study, the RMSE values and forecast biases were calculated from the raw BSE Sensex and Tech Mahindra values because of the fact that different input windows (and thus different values to normalize against) were used to produce the results for each network. Therefore, in order to make a direct comparison of the RMSE values and forecast biases of the DNN and the LSTM RNN, the raw/denormalized values were used to calculate the RMSE and forecast bias. These RMSE values and forecast biases would not be comparable to different data sets, and thus are only useful in this work to directly compare the predictions of the DNN and the LSTM RNN.

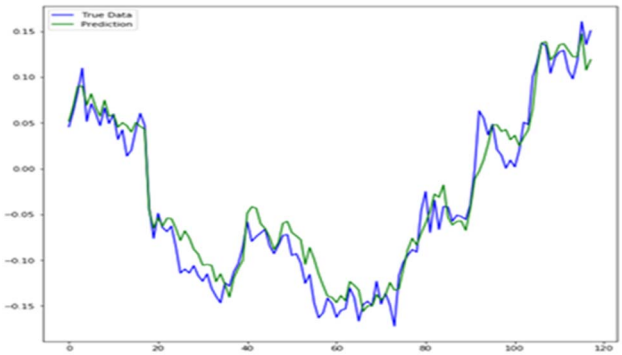

Figure 7b. LSTM daily predictions on NSE: TECHM

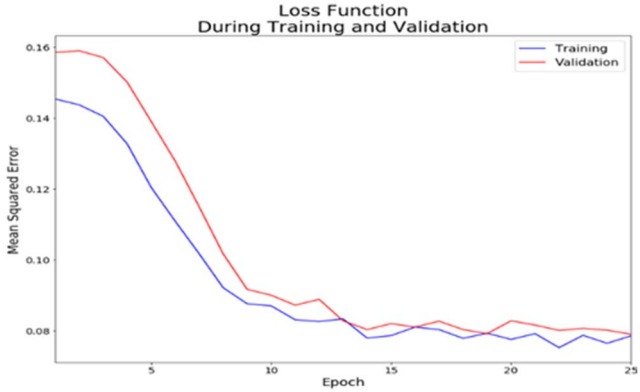

Figure 8a. Training and validation loss of DNN on NSE: TECHM

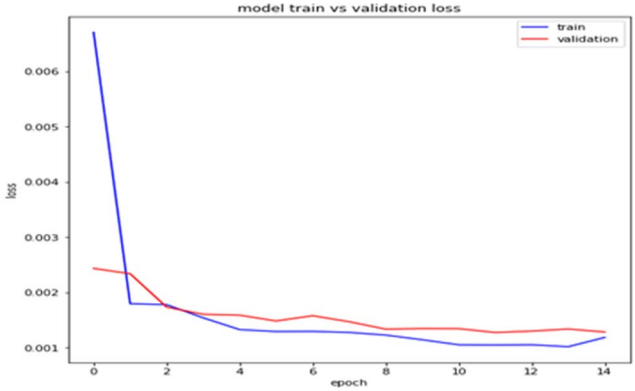

Figure 8b. Training and validation loss of LSTM on NSE: TECHM

D. Improvements

In this study, only price data was used for predictions. It is likely that the inclusion of more impactful features like daily volume, volatility, fundamental ratios etc. would improve the performance of the models. In future studies, the model could also be applied to predict market indices which are developed and less volatile e.g. Nasdaq. Further, analyzing sentiments from news articles and feeding it as an input to the LSTM model along with other features could be an interesting research direction. Lastly, the implementation of more sophisticated optimization techniques could be useful when including more attributes. For example, Bayesian optimization or genetic algorithm could be implemented to optimize hyperparameter selection and to avoid local minima.

VII. Conclusion

We applied the state-of-the-art LSTM and the DNN to stock market price forecasting. We compared results trained on a daily basis for accuracy and error. Both models performed well in making daily predictions. It seems that both the DNN and the LSTM RNN are suitable for this type of task. With the help of strategies to avoid overfitting, both models were able to generalize well to a new, more volatile data set. Finally, the LSTM RNN outperformed the DNN in making weekly predictions. With the inclusion of more attributes, the LSTM RNN shows promise for finding underlying trends and making longer term predictions on volatile stock data sets.

References

[1] Ariyo, A. A., Adewumi, A. O., & Ayo, C. K. (2014). Stock Price Prediction Using the ARIMA Model. In International Conference on Computer Modelling and Simulation, pp. 106–112, IEEE.

[2] Bernal, A., Fok, S. & Pidaparthi, R. (2012). Financial Market Time Series Prediction with Recurrent Neural Networks.

[3] Di Persio, L. & Honchar, O. (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.

[4] Fama, E. H. (1998). Market efficiency, long-term returns, and behavioral finance. Journal of Financial Economics, vol. 49(3), pp. 283-306, Elsevier.

[5] Glorot, X., Bordes, A., & Bengio, Y. (2011) Deep Sparse Rectifier Neural Networks. In International Conference on Artificial Intelligence and Statistics (AISTATS), vol. 15, JMLR:W&CP 15.

[6] Sepp Hochreiter , Jürgen Schmidhuber, Long Short-Term Memory, Neural Computation, v.9 n.8, p.1735-1780, November 15, 1997 [doi>10.1162/neco.1997.9.8.1735].

[7] Kuremoto, T., Kimura, S., Kobayashi, K., & Obayashi, M. (2014b). Time series forecasting using a deep belief network with restricted Boltzmann machines. Neurocomputing, vol. 137, pp. 47–56.

[8] Kuremoto, T., Obayashi, M., Kobayashi, K., Hirata, T., & Mabu, S. (2014a). Forecast chaotic time series data by DBNs. In International Congress on Image and Signal Processing, pp. 1130–1135, IEEE.

[9] Roondiwala M., Patel, H. & Varma, S. (2017). Predicting stock prices using LSTM. International Journal of Science and Research (IJSR), vol. 6(4).

[10] Shen, S., Jiang, H. & Zhang, T. (2012). Stock Market Forecasting using Machine Learning Algorithms. Stanford University.

[11] Singh, R., & Srivastava, S. (2017). Stock prediction using deep learning. Multimedia Tools and Applications, vol. 76(18), pp. 18569–18584.

[12] Zeiler M. D. (2012) ADADELTA: an adaptive learning rate method. arXiv Prepr. arXiv1212.5701.

[13] Wang, J. J., Wang, J. Z., Zhang, Z. G., & Guo. S. P. (2012). Stock index forecasting based on a hybrid model. Omega, vol. 40, pp. 758-766.

[14] Diederik P. Kingma and Jimmy Ba (2014). ADAM: A method for stochastic optimization. arXiv Prepr. arXiv:1412.6980

[15] Overview of mini batch gradient descent. Retrieved May 7th, 2018, from http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.