1. INTRODUCTION

The stock marketplace is a massive array of traders and investors who purchase and promote inventory, pushing the charge up or down. The charges of stocks are ruled through the ideas of call for and supply, and the ultimate aim of purchasing stocks is to make cash through shopping for shares in businesses whose perceived value (i.e., percentage charge) is predicted to upward thrust. Stock markets are carefully connected with the arena of economics —the upward thrust and fall of percentage charges may be traced back to a few Key Performance Indicators (KPIs) [1]. The five most commonly used KPIs are the opening stock price ('Open'), end-of-day price ('Close'), intraday low price ('Low'), intra-day peak price ('High'), and total volume of stocks traded during the day ('Volume') [1].

The economy and stock prices depend primarily on the subjective perception of the stock market. Due to the volatility of factors that play an important role in price movement, it is almost impossible to predict stock prices to the True [1]. However, it is feasible to make an educated estimate of prices. Stock prices never move in isolation: the movement of one stock tends to have an avalanche effect on several other stocks as well [2]. This aspect of stock price movement can be used as an important tool for predicting the prices of many stocks at once. Due to the large volume of money and the number of transactions taking place every minute, there is a trade-off between the accuracy and scope of the predictions made; Therefore, most inventory forecasting systems are distributed and implemented in parallel [3]. These are some of the considerations and challenges in stock market analysis.

With the increase in market size and the speed at which trades are executed, investors are less able to rely on personal experience to discern market patterns. As technology has advanced, investors and researchers have developed many techniques and different models to solve the problems that arise. Examples of such techniques are statistical models, machine learning methods, artificial neural networks, and many more. The first trading procedures generated used historical data from the early 1990s and focused on generating positive returns with minimal risk [4]. In the 2000s, major advances in deep learning and reinforcement learning enabled the development of many hybrid algorithms that use fundamental principles for predicting stock values [4]. However, during the period of the 2008 stock market crash, often referred to as the 2008 Depression, models showed limitations in their ability to make predictions during periods of rapid price changes [5].

As mentioned earlier deep learning techniques can handle uncertainty in the data. In this research, I carried out an analysis using deep learning architecture. The deep learning algorithm encompasses a self-learning method that's capable to identify hidden patterns and dynamics. The stock exchange is non-linear and also the ensuing data is huge. To ascertain this model, and this dynamic data, we want a model to research the hidden patterns and the elementary driving force. The deep learning algorithms are ready to identify and benefit from the interactions and patterns that have information through a self-learning process [1]. In contrast to different algorithms, deep learning mode may be an effective model for these forms of data and may offer a decent forecast analysis of the interaction and hidden patterns within the data.

The deep learning technique carried out in this project is Long-Short Term Memory (LSTM), a type of Recurrent Neural Network (RNN).

General Electric stock price data is used in this analysis. Figure 1 shows the time-series plots of the Open, Close, High, and Low-price data of GE stock price from October 2017 till October 2022 i.e., for 5 years.

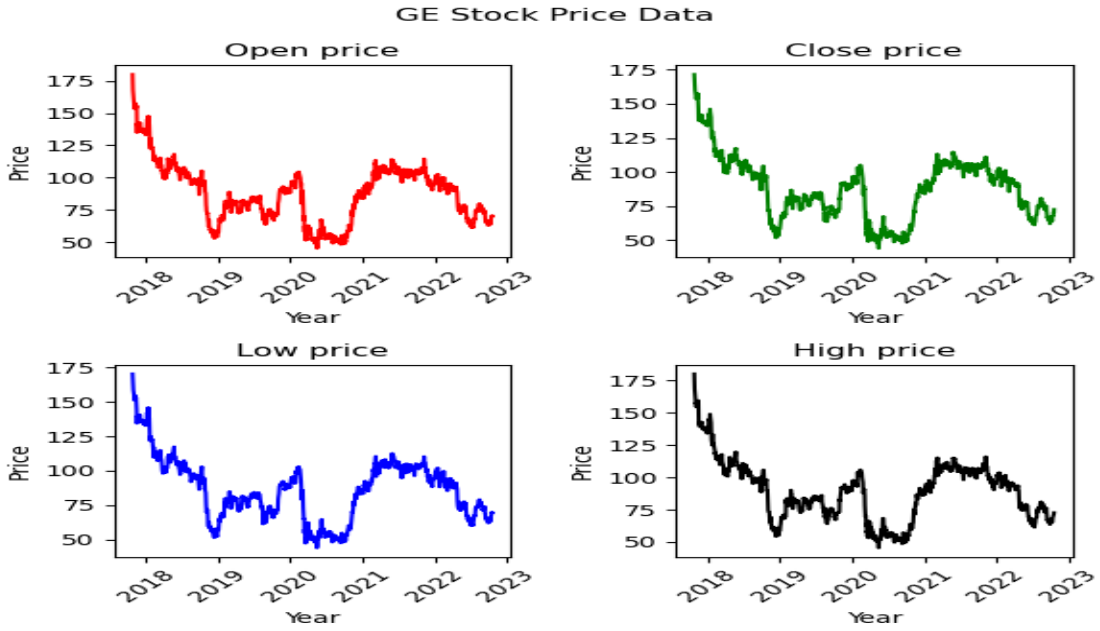


***Figure 1**: Time series plots of Open, Close, Low, and High price data*

In this project, I carried out analysis using different types of LSTM such as single layer LSTM, Stacked LSTM, Bi-directional LSTM, Convolutional Neural Network LSTM, and Multivariate LSTM to determine the most accurate model for the open stock price prediction, for this, we would be accounting RMSE, MAE and R^2 value.

Later on, in this report, we would be discussing the Literature Survey regarding our topic, Systems and their drawbacks, Methodology of Recurrent Neural Networks, Long-Short Term Memory, Advantages of LSTM, Types of LSTM, Description of Data, Analysis, Results, Limitations, and Conclusion.

2. LITERATURE SURVEY

Patel et al. [6] analyzed two ways for adding these species using four distinct speculative models: Vector Support Machine (SVM), Artificial Neural Network (ANN), Random Forest (RF), and Naive-Bayes. The first approach of data entry uses stock trading data to identify ten technical limitations, whereas the second method focuses on recreating these characteristics in the form of static decision data. For each of these two input ways, the accuracy of each speculation model is tested. The research analyzed 10-year historical data from two equities, Reliance Industries, and Infosys Ltd, from 2003 to 2012. Two Indian markets are the S&P Bombay Stock Exchange (BSE) Sensex and the CNX Nifty. According to the test findings, the random field surpasses the other three speculative models in full performance in the first data entry method, 10 technical values are defined as continuous values in this model. According to

the test results, all projected models perform better when these technical frameworks are depicted as data-determining trends. The idea of stock price technical indicators is classified as ‘upwards’ or ‘downwards’ in this paper's Deterministic Trend Data Preparation Structure. Many categories should be evaluated, including “more possibilities to ascend”, “more probabilities to fall”, “fewer probabilities to climb”, “fewer probabilities to fall”, and “medium signal”. This may offer more accurate input into the unreleased professional editing engine, specifically the speculative algorithms of this paper,' explained the researcher. Similarly, Patel et al. [7] published a study in which hybrid models like SVR-ANN, SVR-RF, and SVR-SVR were mentioned [7] showing that the hybrid method outperforms the individualistic models provided in [6].

Chen and Hao et al. [8] look at stock market indicators differently, which is an intriguing and vital lesson in investment and use since the best trading techniques can be most lucrative while minimizing risk. Various methods, including mechanical learning programs, have been developed and tested to produce accurate predictions. This research describes a fully integrated vector weight machine with a K-neighbour feature that can accurately anticipate stock market indications. To begin, develop a complete theory of the SVM data classification feature, which gives different weights for different parameters based on segmentation values. Then, we deduct information from each element's value to acquire weight. They use the K-neighbourhood function in the repositories to forecast future stock market patterns by placing adjacent computers. Finally, the test results of the Chinese stock market indicators known as the Shanghai and Shenzhen stock exchanges were provided to evaluate the efficiency of our work method. Our recommended model can outperform the ‘Shanghai Stock Exchange Composite Index’ and the ‘Shenzhen Stock Exchange Component Index’ in the short, medium, and long term. The approaches provided can also be applied to develop a new financial market indicator.

Chong et al. [9] provided a thorough and objective assessment of the advantages and disadvantages of a thorough investigation of stock market analysis and forecasting approaches. They compared the outputs of three uncontrolled techniques for the release of key component analysis features, autoencoder, and Boltzmann machine bounded to general interconnectivity to predict future stock prices using high-level ‘intraday stock’ recovery as input data: autoencoder, and Boltzmann machine. According to Royal's research, deep neural networks can obtain new information from the leftovers of a self-harming model and increase forecast accuracy; however, this is not the case when a stress model is applied to network residues. Covariance estimates improve greatly when a forecasting function is included in the examination of market structure based on covariance. This study gives valuable information and advice for future research into how extensive research in share market modeling and forecasting, networks can be highly effective. In this work, they provide an in-depth aspect of the stock market prediction models. We develop seven sets of symbols using key components, autoencoders, and Boltzmann's limited range, as well as three Deep Neural Networks (DNN) to anticipate future investment returns ranging from green standards to debt recovery. It was used in the 'Korean stock market,' and you observed that DNNs beat the usual exercise approach in training, but the advantages vanish substantially in testing.

In this work, Nam and Seong et al. [10] provide an innovative research approach for predicting stock price fluctuations based on capital risk assessments. Our method, in particular, looks at the causal links between enterprises and considers the direct impact of the General Divisions of the Industrial Division. The suggested method uses entropy transmission and reads numerous letters to combine the features of the target firm and the causative variable to discover the reason. Based on the external testing of 'The Korean market' data, our test findings suggest that the framework based on causal analysis beats the previous two methods. Furthermore, the test results show that the proposed technique is successful and can effectively anticipate values even when no financial information for the target company is accessible but financial news is available for the causative organizations. Their finding shows that in prediction situations, recognizing causal relationships, as well as constructing machine learning algorithms and establishing well-developed hypothetical links such as sophisticated idea systems, is significant. To increase model accuracy, analyze causal links between non-regulated firms by studying individual companies and within the sector in this study. This strategy enhances prediction by exceeding previous research's critical limit, which is regarded as a dual aim within the GICS regions. To properly examine the causal relationship and apply it to the predictive model, we employ the entropy transmission approach, which is extensively used in physics.

By integrating stock forecasts with later stock market possibilities (including stock forecasts and stock options), Yang et al. [11] developed a two-step mixed stock options approach. (1) Predict future stock trading using computing intelligence (CI) or a high-quality learning machine with strong learning ability and rapid computer speed. (2) The stock exchange system is structured as a mix of predictable line items (formed early on) and assets (renowned in existing books), with stocks listed above the weighted portfolio. Using the Chinese stock market as a research sample, the artistic results show that the novel's sanitary approach, which employs great predictable materials, surpasses traditional methods (excluding stock forecasts), and the same counterparts in outcomes. To better explain the future features of the complicated stock market, the research presents a model for stock options selection based on a mix of stock predictions. The unique model enhances the content in two ways. To begin, it would be the first attempt to merge stock forecasts with stock options to produce a new stock option with predictable qualities to capture impending market structures. Second, it was evaluated in the Chinese stock market against traditional stock options models (other than stock predictions) and comparative partners (in addition to other forecasting models, design characteristics, development algorithms, and robust functions) to assure its efficiency and effectiveness.

J. Long et al. [12] propose an in-depth neural network model that forecasts stock price fluctuations based on idle and open data. Because they rely on other stocks, their strategy leverages an information graph and methods for embedding the graph to identify the optimum target stock for market structure and trading information. Due to the huge number of depositors and the complexities of the job record data, investors are mobilized to reduce the size of matric trademarks, and matriculants are linked to a convolutional neural network to uncover investment styles. Finally, the attention-tracking short-term memory network can predict stock price patterns, which can aid in financial decisions. Price and style indicator analysis outperforms all other estimation methods. The practice's prediction accuracy has reached 70%

or more, which is significantly higher than the previous accuracy of 65%. The model's durability and efficiency were also disclosed by the results.

Yan and Aasma et al. [13] provide a novel technique for learning prediction, which we have matched with a comprehensive model based on the stock-CEEMD-PCA-LSTM study. In this concept, the integration of the integration mode (CEEMD) can be employed as a sequence of declining and lessened modules to discriminate the variance or trend ratios of different time series, giving a variety of internal mode functions (IMFs) of varying sizes. By keeping more information on raw data, PCA reduces the size of the IMF component, eliminates external data, and improves response speed. Following that, high-quality features that were not previously included are individually added to LSTM networks to estimate the closing value of each item on the following trading day. Finally, the sum of the predicted values for each item determines the final predicted value.

Jing, Wu, and Wang [14] presented a mixed model that combines an in-depth learning technique with a stock price analysis model. They employ the Convolutional Neural Network model to distinguish between investors' hidden emotions and those expressed in huge trading volumes. They then devised a hybrid research strategy that used the 'Neural Network's 'Long Short-Term Memory (LSTM)' approach to study market 'Technical Indicators' as well as the outcomes of the previous phase's emotional analysis. Furthermore, the project conducted real-time tests on the Shanghai Stock Exchange (SSE) from six three-time factories to ensure the efficiency and effectiveness of the presented models. The findings also demonstrate that the presented study outperforms the basics in categorizing investor sentiments and that this combined method beats individual models and algorithms that do not take emotions into account when predicting stock prices.

Rezaei, Faaljou, and Mansourfar et al. [15] present a technique based on the idea that integrating these species can improve the analytical power of the model by combining certain algorithms created among them. Empirical research supports this premise, demonstrating that combining CNN with LSTM, CEEMD, or EMD can improve predictive accuracy and outperform other partners. In addition, when compared to EMD, the new CEEMD approach beats it. The primary idea behind the suggested approach was to integrate CEEMD, CNN, and LSTM models to establish interactions between them, which could result in richer features and timelines. An expert algorithm was then used to forecast the stock price.

Lu et al. [16] combine Convolutional neural networks (CNN), Bi-directional Long-term Memory (BiLSTM), and an attention-grabbing (AM) approach. CNN is a feature extraction method that uses input data to extract features. Based on the feature data, Bi-LSTM anticipates stock closing rates for the next day. To increase forecast accuracy, AM is utilized to capture the beneficial effect of the stock closing price on several preceding periods. To demonstrate the effectiveness of this strategy, 1000 Shanghai Composite Index trading days were utilized to forecast the next day's stock close using this and seven other methods. The results reveal that the method's efficacy has increased, with MAE and RMSE at very low values (at 21.952 and 31.694). R-squared is the highest number (its value is 0.9804). When compared to previous methods, the CNN-BiLSTM-AM method accurately predicts stock prices and provides a more dependable platform for investors to make stock investing decisions.

Table 1: *Overview of Literature Review*

Title / Citation	Methods Used	Advantages	Limitations
Predicting stock and stock price index movement using Trend Deterministic Data Preparation and machine learning techniques, Patel et al [6]	Four prediction models ANN, SVM, RF, NB.	When models are evaluated, their performance improves. with deterministic trend data	Only considers short-term forecasting. When technical characteristics are used as continuous values, performance deteriorates.
Predicting stock market index using the fusion of machine learning techniques, Patel et al [7]	Two-stage fusion approach (SVR-ANN, SVR-RF, and SVR-SVR)	The mixed model performs well in comparison to individual forecast methods	No Real-time prediction. Doesn't consider national, or global financial news
A feature-weighted support vector machine and K-nearest neighbor algorithm for stock market indices prediction, Y. Chen and Hao [8]	FWSVN and KNN	Improved short, medium, and long-term forecasting correctness	The good prediction accuracy on the Chinese stock market only. Long-term has higher prediction accuracy compared to short and medium term
Financial news-based stock movement prediction using causality analysis of influence in the Korean stock market, Nam K and Seong N [10]	Multiple kernel learning, Transfer Entropy, and Causal relationship	Even if there is no business news connected to the chosen stock, market values are anticipated.	Historic data is not considered only dependent on financial news
Stock price prediction using deep learning and frequency decomposition, Rezaei, H., Faaljou, H., & Mansourfar, G [15]	EMD-CNNLSTM CEEMD-CNNLSTM	Better prediction accuracy as compared to traditional Hybrid ML models.	Multi-Layer CNN and LSTM models not verified
A CNN-BiLSTM-AM method for stock price prediction, Lu [16]	CNN-BiLSTM-AM	Good prediction and R-squared values	Perform better only on Shangai composite index.
A hybrid model integrating deep learning with investor sentiment analysis for stock price prediction,	CNN, LSTM, CNNLSTM	Outperforms basic categorizing models and the combined model is better than individual models.	Does not take investors' sentiments into account.

Jing, N., Wu, Z., & Wang, H. [14]

3. DRAWBACKS OF SOME EXISTING SYSTEMS

Traditional approaches to stock exchange analysis and stock price prediction embody basic analysis, which looks at a stock's past performance and therefore the general believability of the corporate itself, and applied math analysis, which is exclusively involved with calculation and characteristic patterns in stock price variation. The latter is often achieved with the assistance of Genetic Algorithms (GA) or Artificial Neural Networks (ANN), however, these fail to capture the correlation between stock costs within the variety of long-term temporal dependencies. Another major issue with the exploitation of simple ANNs for stock prediction is the development of exploding/vanishing gradient [17], wherever the weights of an oversized network either become too large or too little (respectively), drastically fastness their convergence to the optimum value. This is usually caused by 2 factors: weights are initialized randomly, and conjointly the weights nearer to the tip of the network also tend to vary a great deal over those at the beginning. An alternative approach to stock market analysis is to cut back the dimensionality of the input data [18] and apply feature selection algorithms to rank a core set of options (such as GDP, oil price, inflation rate, etc.) that have the best impact on stock costs or currency exchange rates across markets [19]. However, this technique does not contemplate long-term trading methods because it fails to take the complete history of trends into account; furthermore, there is no provision for outlier detection. The Hidden Markov Model is accurate only for a really short period and the accuracy of predictions is great on short-term data [20], if the data is for a long period the model performance deteriorates. Even linear regression and multiple regression analysis can be performed on the prediction of stock market price, however, the accuracy produced by these two methods is comparatively less and also does not capture the trends in data exactly.

3.1 Why Long Short-Term Memory (LSTM) is superior to ARIMA in stock price prediction?

Many researchers explain LSTM is superior to ARIMA because of its deep learning architecture and hidden layers in model building, making the model perform better on any kind of data. Even though ARIMA and LSTM are good for time-series data, Root Mean Squared Error and loss on data are less with using LSTM. In [21][22], they performed predictions using ARIMA and LSTM to compare the performance, RMSE, and Mean Absolute Error (MAE) were less and accuracy is high using LSTM [21]. ARIMA predicts the stock price lower than the actual price almost throughout the data, and LSTM predicts the stock price slightly higher than the actual price, however, this is only at a few points explained by Vaibhav et al [22]. ARIMA models are extensively parameterized due to this, they do not generalize well. When applied to a new dataset, a parameterized ARIMA may not produce accurate results. If ARIMA is used for the prediction of stock price, the information given to the investors varies largely with the actual price which may lead to a loss of money or the stock price may not reach their

actual expectation. With the above information, it is concluded that LSTM is the better option for stock price prediction and they also hold information over a long period.

4. METHODOLOGY

This dissertation follows the underlying method proposed by Hochreiter and Schmidhuber in 1997 i.e., Long Short-Term Memory. Long Short-Term Memory (LSTM) networks are recurrent neural networks that can learn order dependence in sequence prediction tasks and are used for deep learning techniques. These methods are used to overcome the long-term dependencies problem associated with Recurrent Neural Networks. The reason for choosing this method is because of drawbacks produced by other methods as discussed in 3 and 3.1. For a detailed understanding, the papers mentioned below can be referred to. Unless otherwise specified, the information in this chapter is derived from the papers listed below. And the methods we are using in this report are Single Layer Long Short-Term Memory, Stacked Long Short-Term Memory, Bidirectional Long Short-Term Memory, Convolutional Neural Networks Long Short-Term Memory, and Multi-variate Long Short-Term Memory. In this chapter, we would be briefing about Deep Learning, Recurrent neural networks, LSTM and its advantages, types of LSTM mentioned above, and hyperparameters.

4.1. DEEP LEARNING

Deep learning is a kind of machine learning and artificial intelligence (AI) that imitates the method humans gain certain kinds of knowledge [12]. Deep learning is a vital part of data science, which incorporates statistics and predictive modeling, it's extraordinarily helpful to data scientists who are tasked with collecting, analyzing, and interpreting large amounts of data; deep learning makes this method quicker and easier [27]. Most of the learning method uses neural network architecture, which is why you would like to review the deep learning model is usually mentioned as a deep neural network. Here, the term "deep" usually refers to the number of hidden layers of the neural network. The traditional neural network consists of only 2-3 hidden layers, whereas deep neural networks with up to 150 hidden layers [24]. These models are trained using large data sets and neural network architecture to learn about features directly from data, eliminating the requirement for human feature extraction. Deep learning methods necessitate a substantial amount of data for training models, as well as a graphics processing unit (GPU) and rapid data processing [8][9].

Deep neural networks have multiple non-linear processing layers, and they use simple elements that run in parallel, similar to the biological neuron system [23]. Deep neural networks are made up of an input layer, a few hidden layers, and an output layer. Every hidden layer receives information from the preceding level through a layer of interconnected nodes or neurons. Based on its inputs from the previous layer, each node determines what to transmit to the next tier.

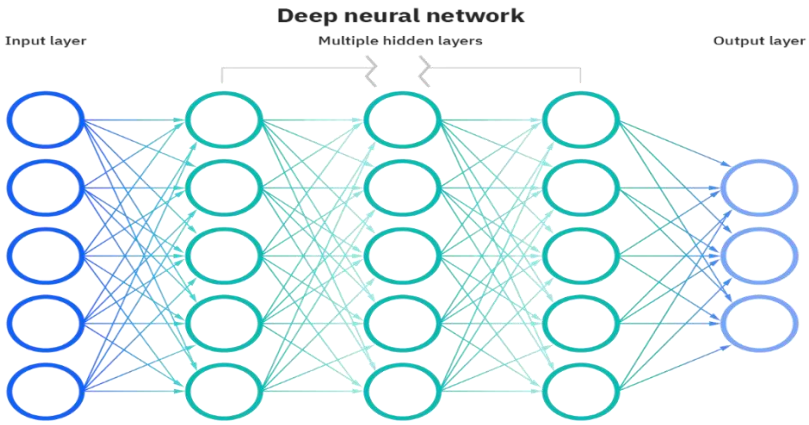


Figure 2: *Deep neural Network Architecture. Image from https://www.ibm.com/cloud/learn/neural-networks*

4.2. RECURRENT NEURAL NETWORKS

Recurrent Neural Networks (RNN) are a category of neural networks specifically designed to handle sequential data. There are two kinds of RNNs, discrete-time RNNs and continue-time RNNs [23]. They are designed with a cyclic affiliation architecture, that permits them to update their current state given the previous states and current input data. RNNs are typically artificial neural networks that encompass standard recurrent cells. These types of neural networks are best known to be accurate in solving problems. It is specialized in processing a chain of values $\chi^1 \ldots \chi^n$ in which n is the total number of features, and χ are the features, inclusive of time-series data [23]. Scaling images with massive widths and heights and processing images of variable lengths is also viable to a huge extent. Furthermore, most RNNs are capable of processing sequences of variable length. In a conventional neural network, all input and output are independent of every other. But for a lot of tasks, it is not a terrible idea. If you need to predict the subsequent word or sentence, you need to realize which phrases come earlier than it. RNNs are referred to as regularly as they perform identical tasks for every element, and the output voltage relies upon the previous word. RNNs have a "memory" that stores the data approximately what has been calculated so far.

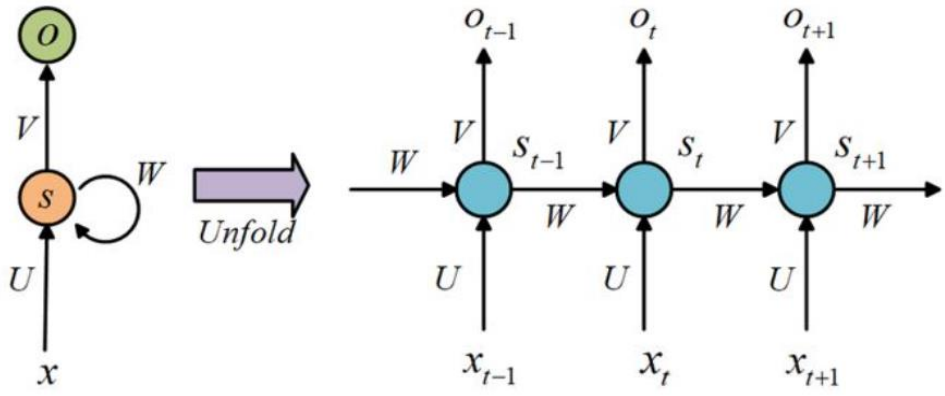


Figure 3*: A RNN diagram of being unrolled full network. Image from https://www.kdnuggets.com/2015/10/recurrent-neural-networks-tutorial.html*

- X_t is the input at time step t.
- S_t is the hidden state a $S_t = f(Ux_t + WS_{t-1})$. This is a "Memory" of the network. S_t calculation is based on the previously hidden and enter in the current step.
- W & U are weighted parameters.
- S_{t-1} is the old state, S_{t+1} is the next state (Predict).
- O_t is the output at time step t.

However, RNNs are missing the capacity to learn long-term dependencies as it is far illustrated in research contacted by Yashoua et al [25]. Therefore, to manage those long-term dependencies, in 1997 Hochreiter and Schmidhuber proposed an answer known as Long Short-Term Memory (LSTM) [26].

4.3. Long Short-Term Memory (LSTM)

Long Short-Term Memory consists of similar RNN architecture that has been developed to outperform RNN on numerical tasks [26]. LSTM has a cell state that contains additional memory which is utilized to store the prediction's relevant previous information. Some of the data is stored in a cell, which represents the status of the transformed structure known as the gate. Normal RNN neurons solely take in their prior hidden state and the current input to produce a new hidden state, however, an LSTM neuron also takes in its old cell state and outputs its new cell state [25].

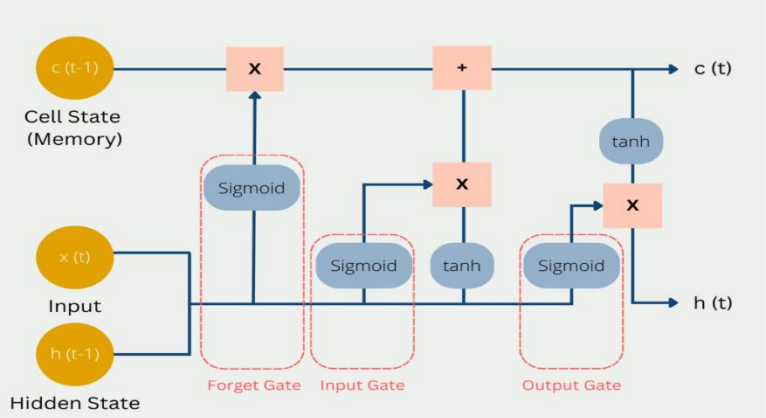


***Figure 4**: LSTM Memory Cell.*

Image from https://databasecamp.de/en/ml/lstms

An LSTM memory cell consists Forget gate, Input gate, and Output gate as illustrated in Fig 4:

Forget Gate: The forget gate determines when particular parts of the cell state should be replaced with more recent data. It returns values near 1 for cell state parts that should be retained and 0 for values that should be ignored.

Input Gate: This portion of the network learns the conditions under which any information should be stored (or updated) in the cell state based on the input (i.e., the previous output h(t-1), input x(t), and prior cell state c(t-1).

Output Gate: This section determines what information is transmitted forward (i.e., output h(t) and cell status c(t)) to the next node in the network based on the input and cell state.

The Sigmoid layer gives numbers between 0 and 1. Tanh Layer produces a new vector that is added to the state. In Figure 4, Vector Operations: **x** is for the scaling of data, and **+** is for the addition of data.

Based on the output of the gate cell's status will be updated. It can be represented mathematically by the below formulas:

- $f_t = \sigma(W_f.[h_{t-1}, x_t] + b_f)$
- $i_t = \sigma(W_i.[h_{t-1}, x_t] + b_f)$
- $\widetilde{c_t} = tanh(W_c.[h_{t-1}, x_t] + b_c)$
- $c_t = f_t * c_{t-1} + i_t * \widetilde{c_t}$
- $o_t = \sigma(W_o.[h_{t-1}, x_t] + b_o)$
- $h_t = o_t * \tanh(c_t)$

Where, f_t: forget gate vector, h_t: output vector, x_t: input vector, c_t: cell state vector, i_t: input gate vector, o_t: output gate vector, and W, b are the parameter matrix and vector.

4.3.1 ADVANTAGES OF LSTM

An LSTM's key advantage is its capacity to learn context-specific temporal dependency [1]. Without explicitly applying an activation function within the recurrent components, each LSTM unit recalls information either for a long or short duration of time (therefore the name). It is vital to remember that any cell state is multiplied only by the forget gate output, which ranges between 0 and 1. In other words, the forget gate in an LSTM cell controls both the weights and the activation function of the cell state [4]. As a result, instead of growing or decreasing exponentially at each time step or layer, information from a prior cell state can travel through a cell unchanged, and the weights can converge to their ideal values in a reasonable time frame. This enables LSTMs to confront the issue of vanishing gradients - because the value stored in a memory cell is not iteratively updated, the gradient does not vanish when trained using backpropagation.

Furthermore, when compared to other RNNs, LSTMs are particularly insensitive to gaps (i.e., temporal lags between input data points).

4.4 PREDICTION MODELS

In this research, we are building 5 different prediction models as discussed earlier. Now, we would be explaining each prediction model and its working in detail.

4.4.1 Single Layer Long Short-Term Memory

The Single Layer LSTM model is a simple LSTM model in which input is fed into an LSTM at each stage. On every time step, the memory cell, i.e., the LSTM memory cell, accepts input such as X_0 at the zeroth position, X_1 at the first position, and so on, and generates output $\hat{Y}_0$, $\hat{Y}_1$, $\hat{Y}_2$, $\hat{Y}_3$…This memory, on the other hand, provides state vectors H_0, H_1, H_2... It is termed a Recurrent Neural Network because this state vector is given as input to the next LSTM cell at each step. This state vector is a replica of the outputs $\hat{Y}_0$, $\hat{Y}_1$, $\hat{Y}_2$, $\hat{Y}_3$ …

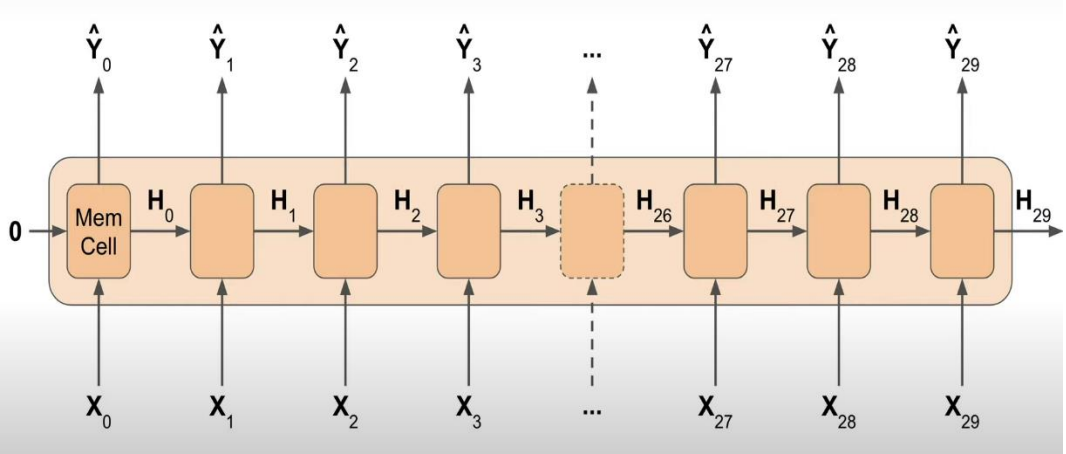


Figure 5: *Single-Layer LSTM architecture. Image from https://www.udacity.com/course/intro-to-tensorflow-for-deep-learning--ud187*

In Fig 5 Mem cell is an LSTM memory cell, X_0, X_1, X_2… are inputs at each time step, $\hat{Y}_0$, $\hat{Y}_1$, $\hat{Y}_2$, $\hat{Y}_3$… are outputs and H_0, H_1, H_2... are state-vectors. In Fig 5, it is shown as just 29 memory cells, however, in actuality, it depends on the number of data points available in the data. And even in our analysis, we have more data points, not just 29.

4.4.2 Stacked Long Short-Term Memory

Graves et al. introduced stacked LSTMs as a reliable strategy for confronting different sequential challenges in their study on speech recognition in 2013 [28]. Existing research [22] has demonstrated that LSTM architectures with multiple hidden layers can build up a higher degree of representation of sequential data, allowing them to perform more effectively and accurately. Its design is made up of numerous stacked LSTM layers, with the output of one model's hidden layer fed directly into the input of the next hidden layer as shown in Fig 6. Stacked LSTM gives a succession of values rather than the usual multilayer LSTM architecture in which a single layer provides a single output value to the subsequent layer.

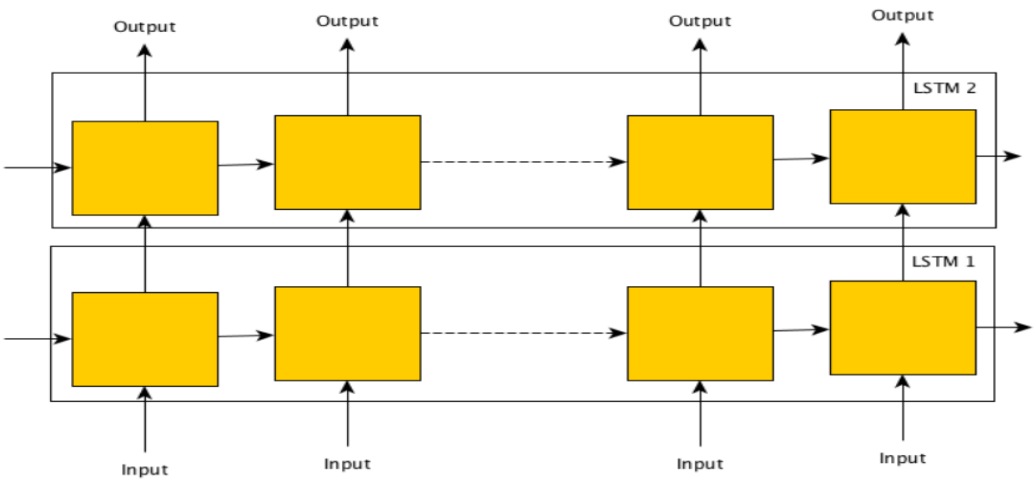


Figure 6:*Stacked LSTM architecture*

4.4.3 Bi-directional Long Short-Term Memory

A Bi-directional LSTM (BiLSTM), developed by Schuster and Paliwal [29], is capable of being trained with data sequences both forward and backward into two independent recurrent networks that are linked into the same output layer. The concept is to divide the state of neurons in a network into two parts: one for forward states beginning with t=1 and another for backward states beginning with t=T [18]. It's also an effective tool for simulating the sequential relationships between words and phrases in both directions.

To summarise, BiLSTM adds an LSTM layer that reverses the direction of information flow as shown in Fig 7. In essence, it means that the input sequence is reversed in the additional LSTM layer. Due to this, it can capture the data that is ignored by normal LSTM. The outputs of both LSTM layers are then combined in a variety of methods, including average, sum, multiplication, and concatenation.

In this BiLSTM inputs are given to the both forward and backward layers at each time step. These layers together are used to generate outputs and return sequences.

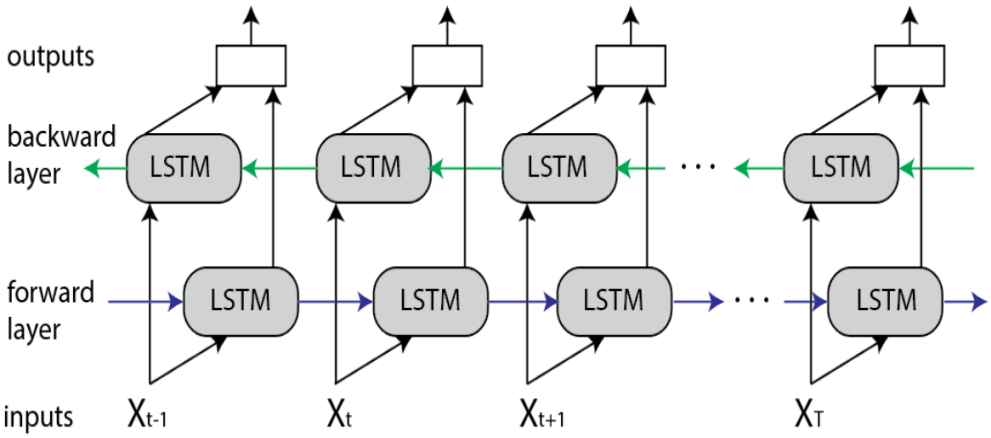


Figure 7: *Bi-directional LSTM architecture.*

Image from https://www.baeldung.com/cs/bidirectional-vs-unidirectional-lstm

BiLSTM will produce a unique output for each component of the sequence. As a result, the BiLSTM model can help with NLP tasks including sentence classification, translation, and entity recognition. It is also employed in speech recognition, protein structure prediction, handwriting recognition, and other domains. However, BiLSTM takes a longer time for training the model when compared to normal LSTMs, so it is advised to use this model only if necessary.

4.4.4 Convolutional Neural Network LSTM

A one-dimensional CNN is a CNN model with a convolutional hidden layer that works on a one-dimensional sequence. In some circumstances, such as with very long input sequences, this is followed by a second convolutional layer, and then a max pooling layer whose purpose it is to condense the output of the convolutional layer to the most salient parts [27]. Pooling that selects the maximum element from the region of the feature map covered by the filter is known as max pooling. As a result, the output of the max-pooling layer would be a feature map that contained the most prominent features of the prior feature map [32]. Pooling happens based on pooling size, in max pooling, if the pooling size is 2, then this layer selects two values at once and returns the maximum value in that size, which can be more understood in Figure 8.

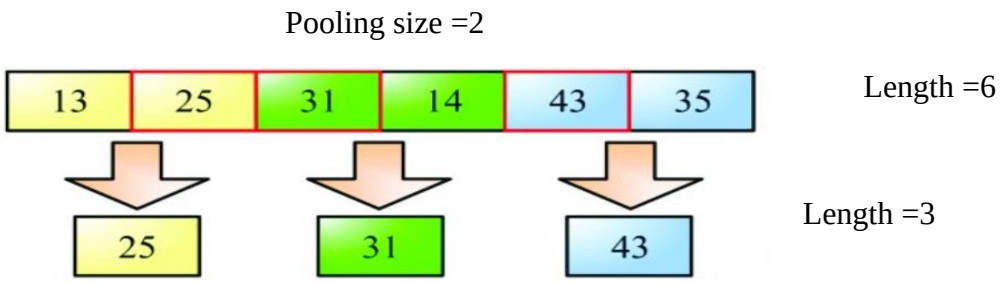


Figure 8: *Max Pooling Sample*

Following the convolutional and pooling layers, there is an LSTM layer and a dense fully connected layer that interprets the characteristics retrieved by the model's convolutional component. To reduce the feature maps to a single one-dimensional vector, a flattened layer is utilized between the convolutional layers and the LSTM layer.

In this CNN LSTM input enters the CNN 1D layer initially, then max pooling is performed on the output of the CNN layer. Later on, we flatten the data, which is used to combine all of the 1-Dimensional arrays generated by pooling feature maps into a single long continuous linear vector, thereby feeding the output to the LSTM layer then the process is carried out similarly to normal LSTMs.

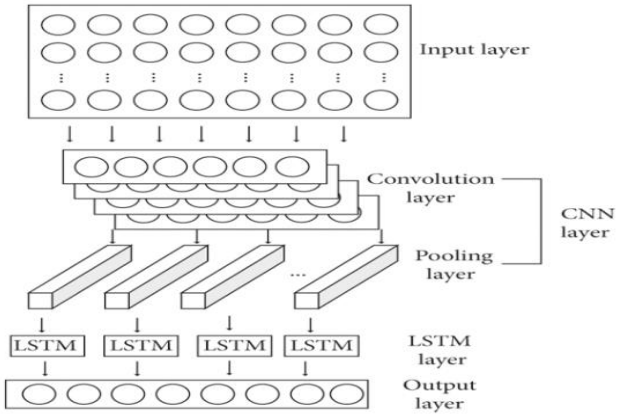


Figure 9: *CNN LSTM Architecture.*

Image from https://www.hindawi.com/journals/complexity/2020/6622927/

4.4.5 Multi-Variate Long Short-Term Memory

A multivariate model is a statistical technique that predicts outcomes by employing multiple factors [15]. Multivariate models are used by research analysts to estimate investment results in various situations to evaluate a portfolio's exposure to specific risks [1]. This enables portfolio managers to better reduce the risks indicated by the multivariate modeling analysis. The process carried out in all the above-discussed models is a univariate model i.e., by feeding only the open price of the stock to the model and making predictions. In the Multi-Variate LSTM model, we make use of all the available prices such as Open, Close, High, Low, and Adjustable Close of stock price data, and make predictions for the Open price of the stock. Multivariate models are useful in many industries to predict their profit, loss, net worth, sales, and so on. In the stock price prediction, this multivariate LSTM is useful, because the open price of the stock is completely dependent on the close, low, high, and adjustable close price of the previous day's price. So, building multivariate LSTM model help in predicting better results for the open stock price.

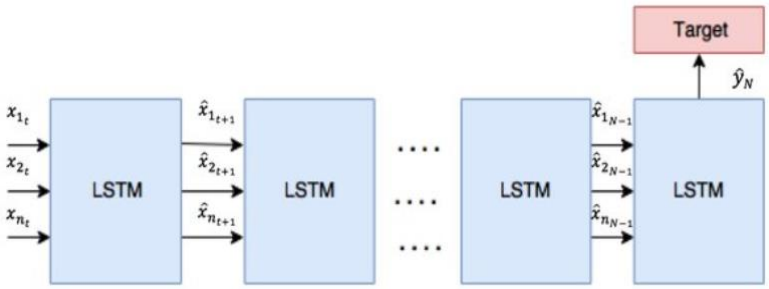


Figure 10: *Multivariate LSTM architecture.*

Image from https://www.researchgate.net/publication/324600237

4.5 HYPERPARAMETERS

Hyperparameters are the factors that influence the network architecture and how the network is trained. These are set before model training.

4.5.1 Units/ Neurons of LSTM

The dimension of the hidden state (or the output) is indicated by the number of units. In figure 11, for example, the hidden state (the red circles) has a length of 2. The number of neurons connected to the layer carrying the concatenated vector of hidden state and input is represented by the number of units (the layer holding both red and green circles below). Two neurons are connecting to that layer in this case.

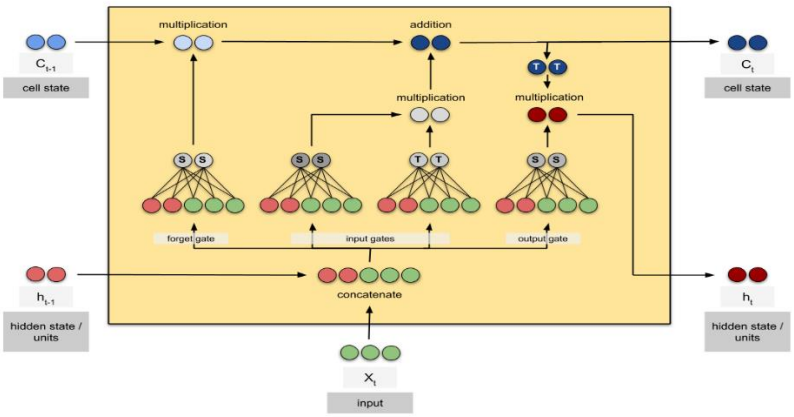


Figure 11:*LSTM cell with units. Image from https://towardsdatascience.com/counting-no-of-parameters-in-deep-learning-models-by-hand-8f1716241889*

The dimension of hidden states (or outputs) and the number of parameters in the LSTM layer are defined by the number of units. Generally, I feel that more units (larger hidden state dimensions) will aid the network in remembering more complex patterns. We can see that the LSTM layer is essentially a feed-forward network with some additional Nonlinear products and matrix operations.

Method: model.add(LSTM(units= 32,……..))

4.5.2 Epochs

Epochs specify how many complete repetitions of the dataset are to be executed. This number can potentially be set to an integer value between one and infinity, it should be increased until validation accuracy begins to drop even while training accuracy increases (and hence risking overfitting).

Method: model.fit(epochs=50, ……..)

4.5.3 Batch Size

Batch size specifies the number of samples to be worked on before the model's internal parameters are updated. For the same number of "seen" samples, larger sizes provide larger gradient steps than smaller ones. A suitable default setting for batch size is 32, which is universally acknowledged. We can experiment with multiples of 32, such as 64, 128, and 256.

Method: model.fit(batch_size=32, ……..)

4.5.4 Dropout

Each LSTM layer should be followed by a dropout layer. By passing randomly selected neurons, such a layer helps to avoid overfitting in training by limiting sensitivity to specific weights of individual neurons. Dropout layers can be utilized with input layers but not with output layers since they can mess up the model's output and error calculation. While boosting complexity may increase the danger of overfitting (by increasing the number of nodes in dense layers or adding more dense layers with poor validation accuracy), this can be mitigated by including dropout. Although 20% is a decent beginning point, the dropout value should be kept low (up to 50%). The 20% threshold is commonly regarded as the optimal compromise between eliminating model overfitting and maintaining model accuracy.

Method: model.add(LSTM(units= 32,……..))

model.add(Dropout(0.2))

4.5.5 Activation functions

It's just a simple function that you use to obtain the output of the node. It is also referred as the Transfer Function. It is used to determine the yes or no output of neural networks. It converts the resulting values from 0 to 1 or -1 to 1, and so on (depending upon the function). There are two types of activation functions.

1. Linear Activation Function
2. Non-Linear Activation Function

Linear Activation Function: This activation function is linear, and the output of this function is not restricted between any range. It does not affect the complexity or other properties of typical data that is given to neural networks.

Equation: $f(x) = x, \quad for(-\infty, \infty)$

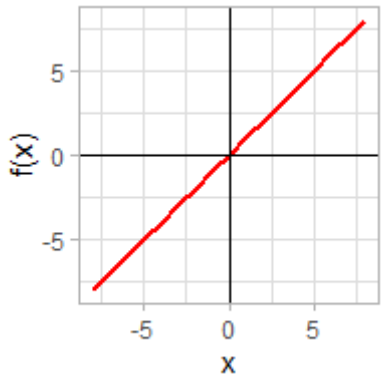


***Figure 12**: Linear Activation Function*

Non-Linear Activation function: This type of function is a widely used activation function; it makes simple for the model to generalize or adapt to a wide range of data and differentiate between outputs.

Non-Linear Activation functions are divided concerning their ranges or curves. They are the Sigmoid activation function, Tangential activation function, RELU activation function, Leaky RELU, and Exponential Linear Unit function. Here in this analysis, we are using only RELU because of its characteristics.

RELU activation function: Rectified Linear Unit (RELU) is an element-wise operation that replaces all received negative values to zero and positive values remain same. This function is either completely non-increasing or non-decreasing, this kind of function is known as monotonic functions.

Equations: $f(x) = 0, \quad for\ x < 0$

$$f(x) = x, \quad for\ x \geq 0$$

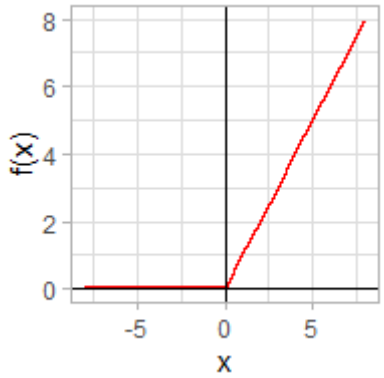


Figure 13: *RELU activation function*

4.5.6 Early Stopping

Early Stopping is to track the loss on the validation set during the training phase and use it to determine when to stop training such that model is accurate but not overfitting.

4.5.7 Learning Rate

Learning Rate (LR) explains how quickly the network's parameters are updated. Setting a high learning rate speeds up learning, but the model may fail to converge or even diverge. A smaller rate, on the other hand, will significantly slow down learning since steps towards the minimum of the loss function will be minimal, but will allow the model to converge gradually.

Method: ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=10, verbose=1)

- Monitor: quantity to be monitored
- Factor: the factor by which the learning rate will be reduced
- Patience: number of epochs with no improvement after which learning rate will be reduced.
- Verbose: int. 0: quiet, 1: update message

4.5.8 Model Checkpoints

The ability to save and retrieve a model's state is important for a variety of applications, including transfer learning and inference using pre-trained models. Saving a model's parameters (weights, biases, etc.) in a checkpoint file or directory is one method [30]. This includes a high-level interface for loading and storing checkpoints in TensorFlow v2 format, as well as lower-level components for writing to and reading from this file format.

5. DATA

The data set is obtained from finance.yahoo.com [31]. Yahoo is one of the top stock research resources because it is free and gives stock data from all over the world. Yahoo provides General Electric stock prices from 1972 to 2022, in our analysis, we are using data from 2017 to 2022 i.e., the past 5 years which has approximately 1260 records of each feature. The head snippets of the data set are given in Table 2. All the prices mentioned are in USD currency.

Table 2: *Top 5 rows of Research data frame*

Date	Open	High	Low	Close	Adj Close
2017-10-23	180.307693	180.307693	170.615387	171.692307	163.258240
2017-10-24	170.153839	171.615387	167.307693	168.384613	160.112991
2017-10-25	168.000000	168.384613	163.846161	165.384613	157.260376
2017-10-26	165.923080	166.538467	163.076920	164.000000	155.943741
2017-10-27	163.538467	163.846161	158.769226	159.923080	152.067123

The data set represents daily OHCL means (Open High Close Low) and Adjustable close. “Open Price” refers to the price at which stock started on a respective day. “Close Price” refers to the completion of the stock price on a respective day. The terms “High” and “Low” refers to the prices of a certain stock at its highest and lowest values in a given day, respectively. The daily closing price solely reveals information about a stock's cash component. It is the price at which the last lot of stock was purchased or sold in the previous trading session. The adjusted closing price takes into account all of the factors that may have affected the stock price after market hours, including corporate activities such as dividends, stock splits, and rights sales. Some of the stock price datasets are also given in form of OHCL diagrams as shown in figure 14.

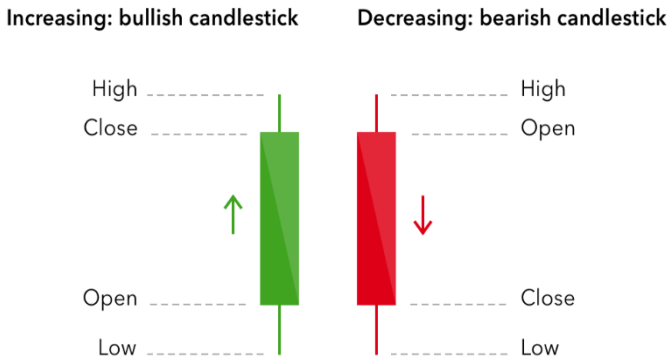


Figure 14: *Candle Stick diagram for OHCL.*

Image from https://www.ig.com/uk/trading-strategies/candlestick-trading-explained-181218

5.1 DATASET BASIC STATISTICS

To understand the basic statistics such as mean, standard deviation, minimum and maximum, we use NumPy and pandas libraries from python.

Table 3: *Basic Statistics of Dataset*

Feature	Mean	Standard deviation	Minimum	Maximum
Open	87.687	22.668	44.88	180.307
Close	87.549	22.524	43.92	171.692
High	88.962	22.686	45.28	180.307
Low	86.286	22.498	43.84	170.615
Adjustable Close	86.217	21.442	43.46	163.258

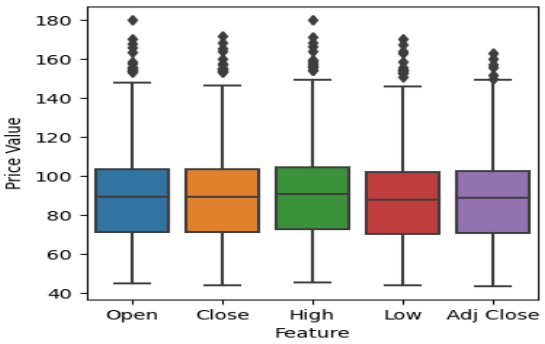
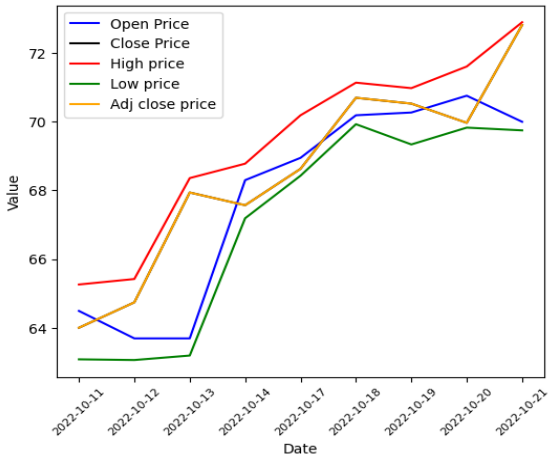


Figure 15: *Boxplot of features in dataset*

Table 2 shows the statistics and Figure 15 shows the boxplot of the features of the data. From this, we can conclude that the Mean, Standard deviation, Minimum and Maximum values of all features are almost near with minimal difference, which makes the features highly correlated. Furthermore, Figure 16 shows the trends of features for the last 10 days in data and Figure 17 shows the correlation value of features, thereby supporting that features are highly correlated.



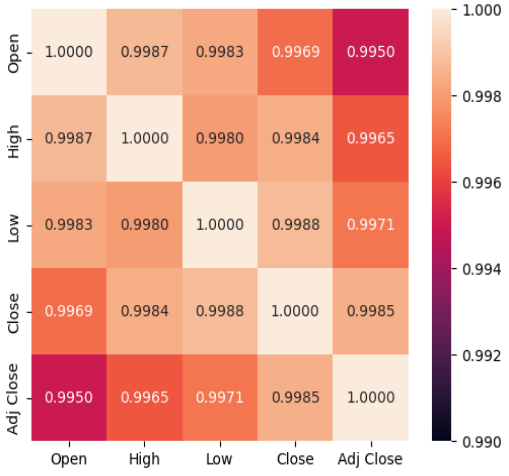


Figure 16: *Stock Price data for last 10 days*

Figure 17: *Correlation heatmap of data*

From above figure 16, it can be concluded that Open and Close stock price data mostly falls in between High and Low stock price data. In this figure, we can also notice that the adjustable close and close price is equal for the last 10 days.

5.2 DATA PRE-PROCESSING

Data pre-processing is the process of transforming raw data into a useful, understandable format. This process is carried out on machine learning models to achieve the best data analysis and good accuracy of models. Data pre-processing of 4 types: Integration, cleaning, Reduction, and transformation. However, we need not perform all types of processing, it is advised to perform all the above 4 types when the data is demanding. In our analysis, we would be performing Reduction and Transformation.

5.2.1. Dimensionality Reduction

Dimensionality Reduction is also known as dimension reduction used to reduce the number of features or input variables of a dataset. In this report we are predicting the open price of stock price, however, we have a dataset that consists of Open, Close, High, Low, and Adjustable close of stock price. So, while performing analysis for single-layer LSTM, Bi-directional LSTM, CNN LSTM, and stacked LSTM, the process carried out is univariate time analysis. For these models, we need to reduce the dimensions of the original data to reduce the cost and time of analysis. Principal component analysis (PCA) is one of the methods used for dimension reduction, in this analysis PCA is applied to extract the open value of stock price to carry out analysis and predictions. And for Multivariate LSTM we would be feeding the model with all the available variables, so it does not require Dimension reduction.

5.2.2. Data Transformation

The process of transforming data from one format to another is known as data transformation. In essence, it includes strategies for translating data into acceptable forms from which the model may efficiently learn. There are many techniques for data transformation they are: Smoothing, Normalization, Aggregation, Generalization, Discretization, Feature Construction, etc., In our analysis, we would be transforming data using normalization. Normalization is the process of transforming original data into a specific range, for example, 0 to 1. The Min-Max Scaler method is used to normalize our data in the analysis. And this process is to be carried out for all the models and reduces bias in the data.

5.3 SPLITTING DATA

To evaluate the model performance, we would be splitting our data into the Train and Test dataset, this is performed to minimize the effect of data discrepancies. In our analysis, we are splitting data into 75% training data and 25% test data. As we are aware that training data is used to train the model and thereafter we test the model by predicting against test data.

6. IMPLEMENTATION OF MODELS

This chapter shows how the theoretical framework given in Chapter 4 and Chapter 5, can be used to develop an actual model. Initially, we would be discussing an overview of the model followed by Single-layer LSTM, Stacked LSTM, Bi-directional LSTM, CNN LSTM, Multivariate LSTM model analysis, their parameters, and hyperparameters in bringing the optimum model

6.1 OVERVIEW OF THE MODEL

The modeling process is divided into six steps as shown in Figure 18. First and foremost, providing the necessary inputs is obtained as explained in chapter 5. Second, the data is then processed as elaborated in 5.2 and prepared for the following steps. Thirdly, the data is split into 75% training and 25% testing data sets as in 5.3. Fourth, train the model with a training data set and evaluate it with loss (Mean Squared Error). Finally, forecasting the Open stock price for the available data set dates and forthcoming 100 days.

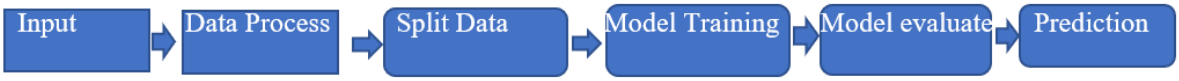


Figure 18: *Flowchart of the process*

6.2 SINGLE LSTM MODEL ANALYSIS

This model works as explained in 4.4.1 and as discussed in 5.2 we would be carrying data processing to extract only the open price of the stock and normalize the data to reduce the bias in the data, then we split data as described in 5.3. Now the data is ready for modeling. However, in building LSTM models we need many hyperparameters as shown in 4.5. Initially, we would be setting the hyperparameters (explained in 4.5): Units =64, Batch size = 32, Dropout =0.2, Optimizer = 'adam', epochs = 50, Linear activation function for the dense. Callbacks such as Early stopping (4.5.6) and Learning rate (4.5.7) are determined based on the validation loss after each epoch. Then we would be fitting the model with training data and give validation data as test data. After evaluating the model, we get a loss i.e., the Mean Squared Error of the model. And there would be only one LSTM layer in this model followed by dropout and dense layers. The single-layer LSTM model built with the above hyperparameters might or might not be the optimum model. To get the optimum single-layer LSTM model, we need to tune the hyperparameters by giving three different values for Units = [16,32,64], Batch size = [32,64,128], Optimizers = ['adam', 'adamax', 'SGD', 'RMSprop'], epochs= [20,50,100], and giving same value for dropout = 0.2 as it is widely recognized as the best value. With all the above hyperparameters we would be performing a cartesian product to get all the possible combinations of hyperparameters.

Number of combinations = number of units parameter values* number of batch size values* number of optimizers* number of epochs values* number of dropout values

$$= 3*3*4*3*1 = 108$$

Then we would be iterating the model by fitting the model and evaluating its results with all possible combinations of hyperparameters and storing the hyperparameters and results of each combination in a list. Later we would be converting this list to a data frame and arranging the data frame based on the validation loss in ascending order. Then we would be selecting the only top 5 snippets because going deep might have a high loss rate. Table 4 shows the top 5 best combinations.

Table 4: *Top 5 combinations of hyperparameters for Single LSTM*

Sl.No	Units	Batch size	Dropout	Optimizer	Epochs	Train data MSE	Test data MSE
1	64	64	0.2	RMSprop	50	0.000327	0.000176
2	64	128	0.2	RMSprop	50	0.000353	0.000180
3	64	64	0.2	RMSprop	100	0.000336	0.000181
4	32	64	0.2	RMSprop	100	0.000357	0.000185
5	64	128	0.2	Adam	100	0.000358	0.000186

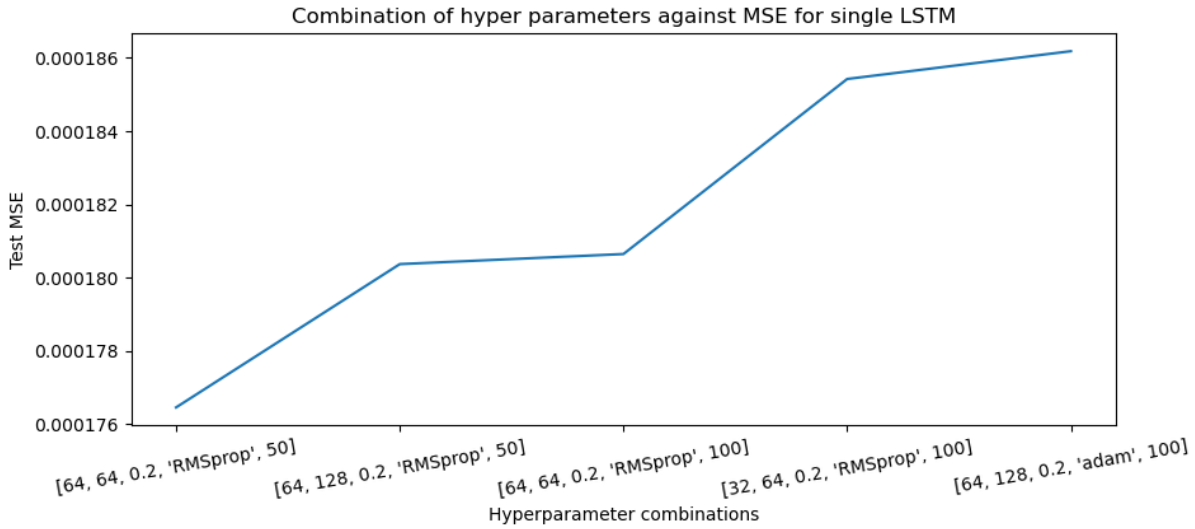


Figure 19: *Hyperparameters combinations against Test MSE for Single LSTM Layer*

From Table 4 and Figure 19, we assume that the 3rd hyperparameter combination i.e., Units=64, Batch size =64, dropout=0.2, optimizer= 'RMSprop', epochs = 100 are the best combination because going for the first two combinations might lead to overfitting and going for the later combinations might result in the high loss rate. And calculating improvement percentage comparing before and after tuning hyperparameters model results.

$$Improvement = \frac{Results\ before\ tuning - Results\ after\ tuning}{Results\ before\ tuning} * 100 \rightarrow (equation\ 1)$$

Table 5: *Results before and after tuning hyperparameters for Single LSTM in normalized scale for Test data MSE*

Results before tuning Test set MSE	Results after tuning Test set MSE	Improvement (%)
0.000183	0.000181	1%

From Table 5, we notice there is a 1% improvement in the results after tuning, we would be modeling with best combinations, and predicting the train data, test data, and 100 future days.

Then we would be de-normalizing the prediction values and test and train data values. Later with the help of train and test predictions and actual data, we calculate residuals which in turn helps calculate forecasting error to get 95% Prediction intervals. Figure 20, shows the predictions using a Single LSTM analysis, it predicts the data well and captures the trends in the data. And almost 90% of the predictions lie in Prediction intervals, thereby making it good model.

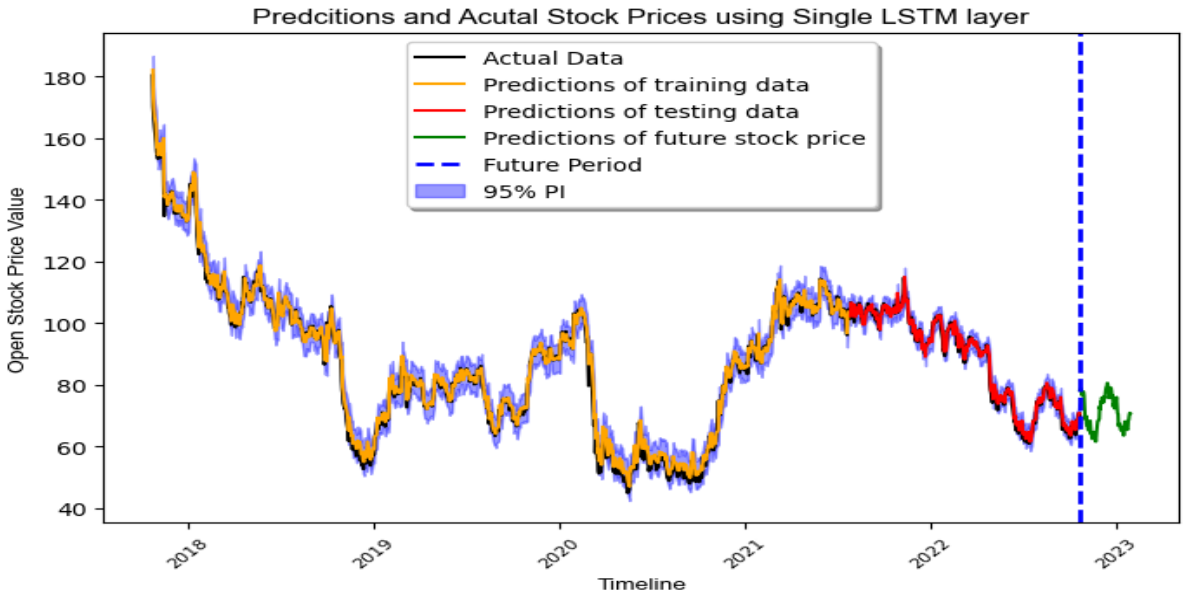


Figure 20: *Actual data and Predictions of open stock price using Single-Layer LSTM*

6.3 STACKED LSTM ANALYSIS

This model methodology is explained in 4.4.2. and we would be having four LSTM layers each layer takes input from the preceding layer. We are carrying a univariate analysis i.e., only for Open stock price so we would be using the already extracted and normalized data used in carrying a single LSTM process, then splitting data as mentioned in chapters 5.2 &5.3. Now setting initial hyperparameters: Units=64, Batch size= 32, dropout = 0.2, Optimizer = 'adam', epochs= 50, linear activation function for dense layer. Callbacks are implemented depending on the validation loss after each epoch. Afterward, we train the model using train data and test data as validation data with initial hyperparameters and then evaluate the results. We cannot conclude that this is the best model as there is room for tuning hyperparameters, now we would be tuning them by giving some set of values for each parameter they are: Units= [16,32,64], Batch size= [32,64,128], Optimizers = ['adam', 'adamax', 'SGD', 'RMSprop'], epochs= [20,50,100] and giving dropout probability as 0.2. With the above values, we would be getting 108 different possible combinations as mentioned in 6.2. Then we would be iterating through all combinations by fitting the model, evaluating it, and storing the train and test loss i.e., MSE. Then we would be arranging the combinations based on test loss in ascending order and from the top 5 combinations we decide the optimum model to avoid both overfitting and underfitting. Table 6 shows the best hyperparameter combinations for the Stacked LSTM model.

Table 6:*Top 5 combinations of hyperparameters for Stacked LSTM*

Sl.No	Units	Batch size	Dropout	Optimizer	Epochs	Train data MSE	Test data MSE
1	64	64	0.2	RMSprop	50	0.000287	0.000163
2	64	128	0.2	Adam	50	0.000298	0.000166
3	64	128	0.2	Adam	100	0.000300	0.000168
4	64	32	0.2	Adam	50	0.000319	0.000170
5	64	32	0.2	Adam	100	0.000298	0.000171

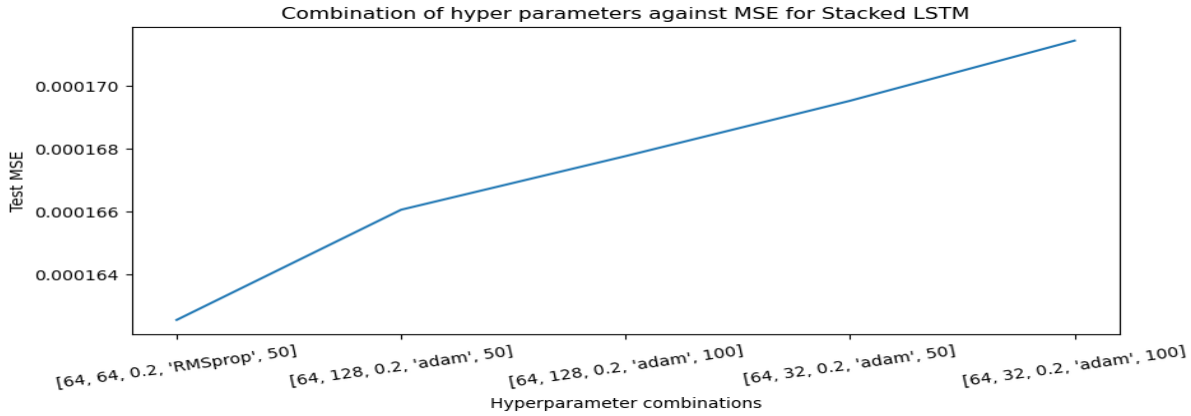


Figure 21: *Hyperparameters combination against Test MSE for Stacked LSTM layers*

From Table 6 and Figure 21, we would be assuming the 2nd combination i.e., Units=64, batch size=128, dropout = 0.2, optimizer= 'adam', epochs=50 is the best combination that does not overfit or underfit the data. Now, we would be comparing its results with initial hyperparameters results to get an improvement percentage using equation 1.

Table 7: *Results of Test set MSE before and after tuning hyperparameters for Stacked LSTM in normalized scale*

Results before tuning Test set MSE	Results after tuning Test set MSE	Improvement (%)
0.000311	0.000166	46%

The Improvement obtained after tuning is 46% which is high, this also concludes that the initial hyperparameter combination is not good for stacked LSTM. Afterward, we build the model with the best combinations and would be predicting test, train data, and 100 future days values and de-normalizing them to get predictions in actual scales. Using the predictions and actual values of test and train data we get RMSE for both datasets and we also calculate residuals to get the forecasting error which in turn helps get 95% Prediction Intervals.

Figure 22, shows the predictions of the train, test datasets, and 100 future days. From figure we can notice that predictions are good and fall in 95% prediction Intervals and trends are also captured in almost similar to actual data.

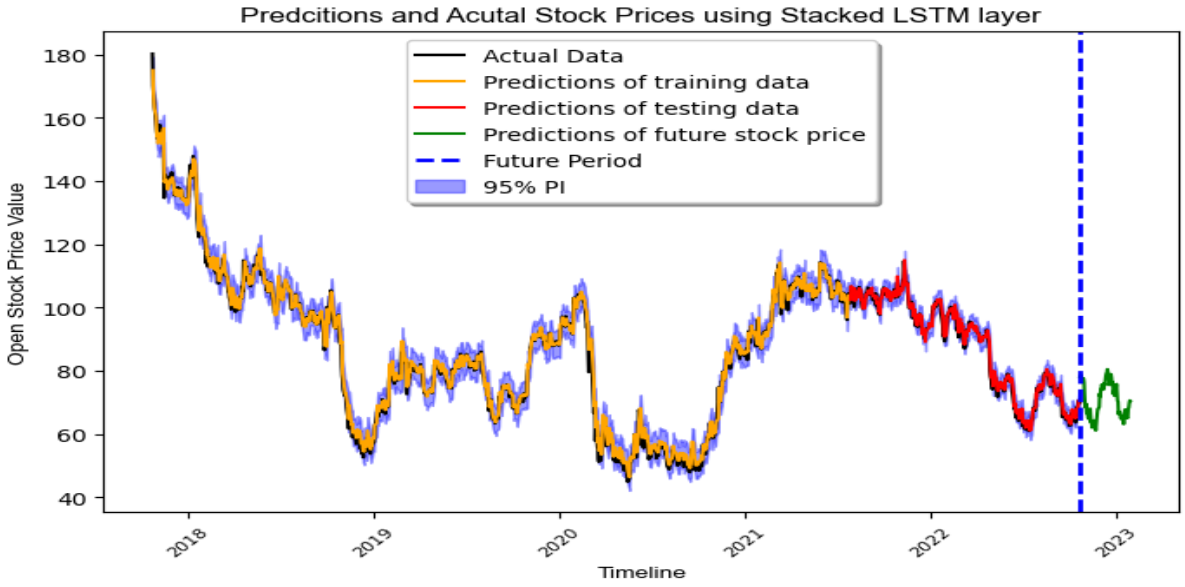


Figure 22: *Actual and Predictions of open stock price using Stacked LSTM*

6.4 BIDIRECTIONAL LSTM ANALYSIS

This model works as explained in 4.4.3, it can predict in both forward and backward directions i.e., using the past it can predict the future, and using the future it can predict the past. This would be having just one bidirectional LSTM layer. Initially, we would be extracting open price data and normalizing the data and then we split the data making it ready for model inputs. Then we set the Initial hyperparameters as Units=64, Batch size =32, Dropout = 0.2, Optimizer= 'adam' and epochs = 50, RELU activation function for the Bidirectional layer and linear activation function for dense layer, Early stopping and Learning rate are determined based on validation loss after each epoch. Now we fit the model and evaluate its results for test data. Then we tune hyperparameters and build models to check if there is any improvement in results. For this tuning we set Units= [16,32,64], Batch size= [32,64,128], Optimizers= ['adam', 'adamax', 'RMSprop', 'SGD'], epochs= [20,50,100] and dropout =0.2. Afterward, we perform a cartesian product to get all possible combinations it would be 108 in total as shown in 6.2. Later we iterate through all combinations, build a model and evaluate the results for each combination. Then arranging the combinations based on test data MSE in ascending order. Table 8 shows the top 5 best hyperparameter combinations using Bidirectional LSTM.

Table 8:*Top 5 combinations of hyperparameters for Bidirectional LSTM*

Sl.No	Units	Batch Size	Dropout	Optimizer	Epochs	Train data MSE	Test data MSE
1	64	128	0.2	RMSprop	100	0.000399	0.000183
2	64	64	0.2	RMSprop	50	0.000345	0.000186
3	64	32	0.2	RMSprop	50	0.000442	0.000204
4	32	128	0.2	RMSprop	50	0.000453	0.000209
5	64	32	0.2	Adam	50	0.000418	0.000211

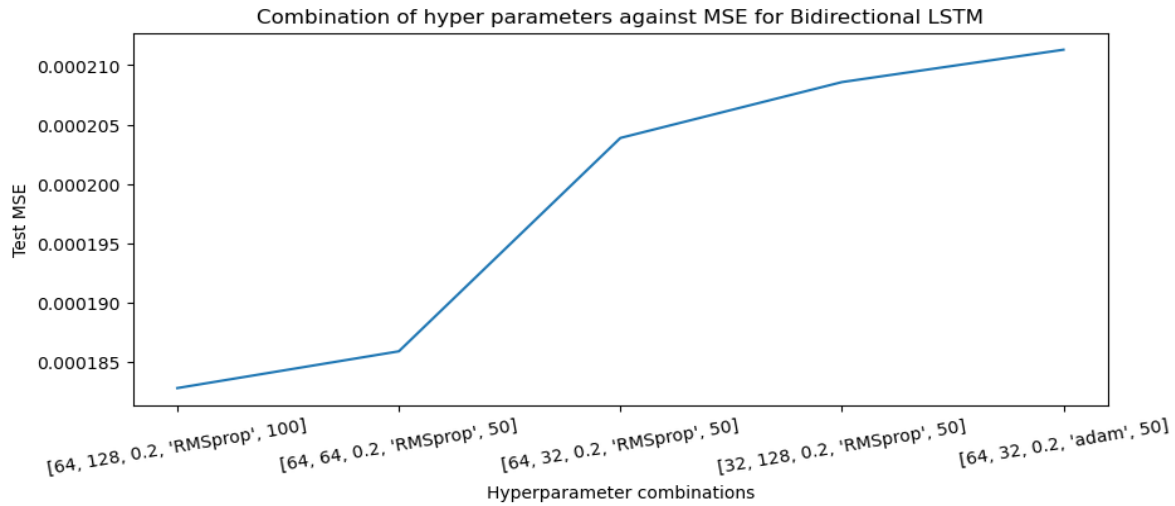


Figure 23: *Hyperparameters combinations against Test data MSE for Bidirectional LSTM*

From table 8 and Figure 23, we assume 2nd combination is the best combination i.e., Units= 64, Batch size = 64, dropout=0.2, Optimizer = 'RMSprop', epoch=50 because 1st combination has MSE too low might result in overfitting and difference in MSE in 2nd and 3rd combination is pretty high. Now we calculate the Improvement percentage with the help of results before and after tuning hyperparameters using equation 1.

Table 9:*Results of Test set MSE before and after tuning hyperparameters for Bidirectional LSTM in a normalized scale*

Results before tuning Test set MSE	**Results after tuning Test set MSE**	**Improvement (%)**
0.000204	0.000186	9%

From Table 9, we notice there is a 9% improvement after tuning hyperparameters, we build the model with the best combination and predict train, test data, and 100 future days, then de-normalize to get data in actual scales. With the help of these train and test predictions, actual data calculate RMSE in the original scale of the data and also calculate forecasting error to get a 95% Prediction Interval.

From Figure 24, we notice that Bidirectional LSTM predictions are good and more than 90% of them fall in the 95% prediction interval and the capturing of trends is also good. From this figure, we can also notice that when there is a huge drop in price i.e., in late 2018 and in 2020, the difference between actual and predicted values would be higher comparatively.

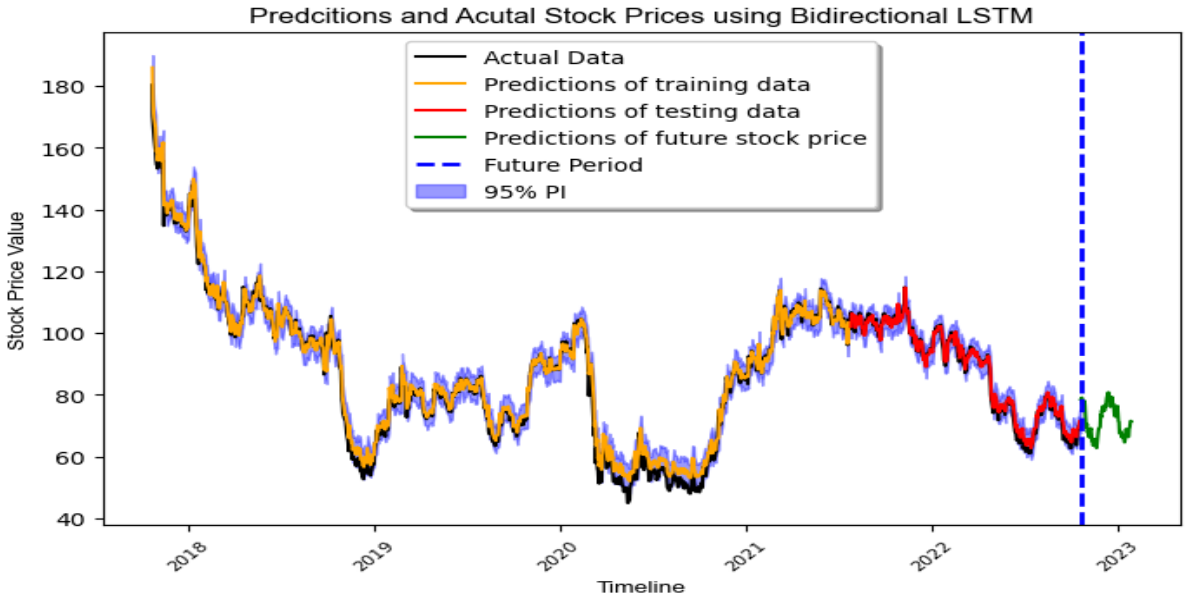


Figure 24: *Actual and Predictions of the open stock price using Bidirectional LSTM*

6.5 CONVOLUTIONAL NEURAL NETWORK LSTM ANALYSIS

This can also be termed CNN LSTM and it works as explained in chapter 4.4.4. In this model we perform data processing i.e., extracting open stock price data and normalizing to reduce any bias in data, splitting data into train and test data, and set initial hyperparameters as Units=64, Batch size = 32, Dropout = 0.2, Optimizer= 'adam', epochs=50 and linear activation function for dense layer. Then we fit the model with initial hyperparameters using train data, having test data as validation data, and evaluating results. Thereafter, we tune hyperparameters by giving different sets of values for each Units= [16,32,64], Batch size= [32,64,128], Optimizers= ['adam', 'adamax', 'RMSprop', 'SGD'], epochs= [20,50,100] and dropout=0.2. With this above set of hyperparameters, we get a total of 108 combinations by performing cartesian product as shown in 6.2. Later we iterate through each of the 108 combinations and build models and evaluate the results for each combination. Arrange the results of every combination in ascending order based on test data MSE. From that, we select the top 5 combinations. Table 10 shows combinations and results.

Table 10: *Hyperparameter combinations and results for CNN LSTM.*

SL.No	Units	Batch Size	Dropout	Optimizer	Epochs	Train data MSE	Test data MSE
1	64	32	0.2	Adam	20	0.000331	0.000166
2	64	32	0.2	Adam	100	0.000336	0.000175
3	64	64	0.2	Adam	20	0.000417	0.000178
4	64	32	0.2	Adam	50	0.000404	0.000181
5	16	64	0.2	RMSprop	100	0.000442	0.000190

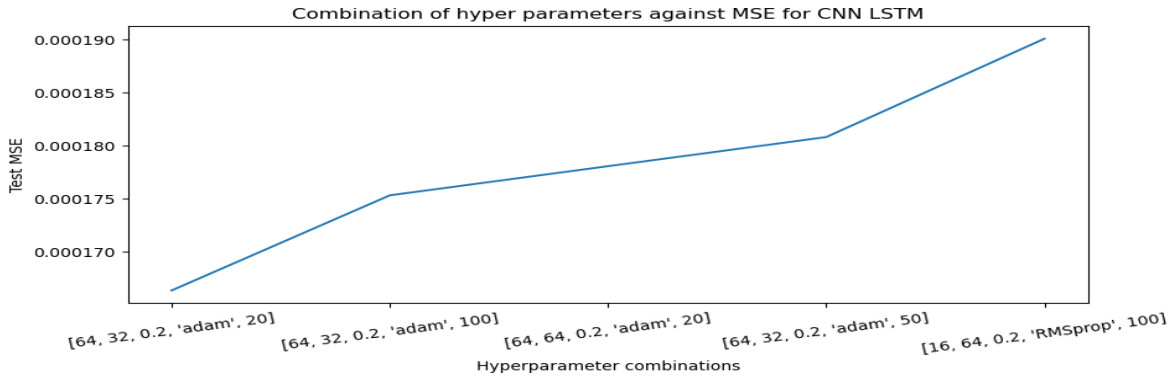


Figure 25: *Hyperparameter combinations against Test MSE for CNN LSTM*

From Table 10 and figure 25, we assume that the 3rd combination i.e., Units=64, Batch size= 64, dropout = 0.2, Optimizer = 'adam', and epochs= 20 is the best combination with the optimum MSE. Now let's calculate the improvement percentage of results before and after tuning hyperparameters using equation 1.

Table 11:*Results of Test set MSE before and after tuning hyperparameters for CNN LSTM in a normalized scale*

Results before tuning Test set MSE	**Results after tuning Test set MSE**	**Improvement (%)**
0.00022	0.000178	19%

We noticed a 19% improvement in the results after tuning from Table 11, so now we need to build the model with the best hyperparameter combinations and predict train, test data, and 100 future days. Then we de-normalize the predictions, later with help of predictions and actual data we find RMSE for training and testing data in actual scales and also forecasting errors to achieve a 95% prediction interval.

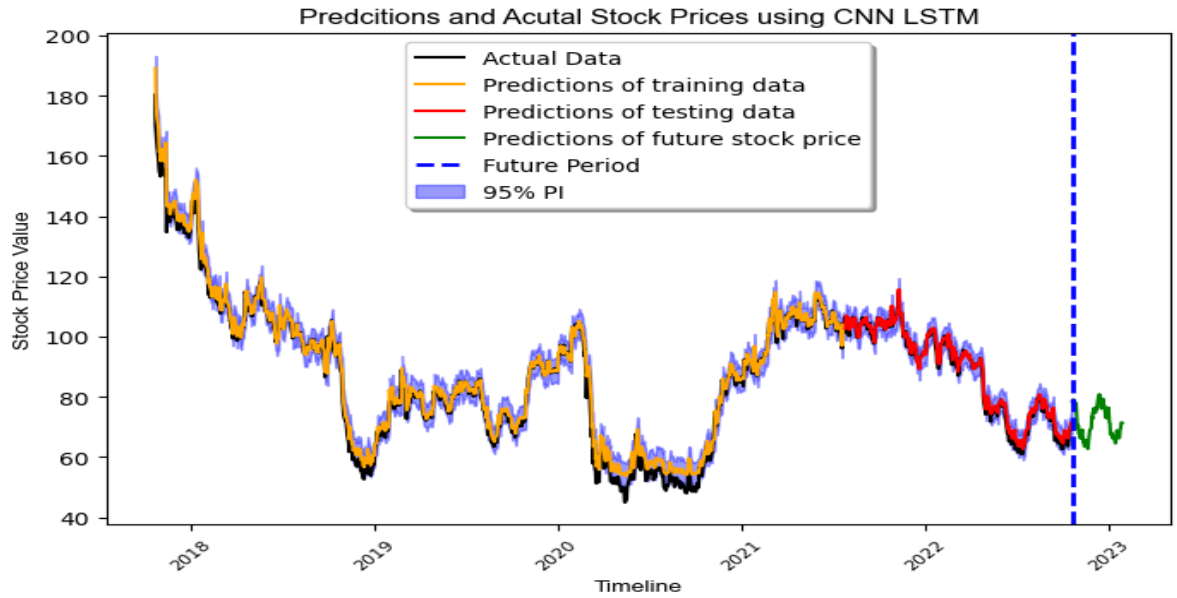


Figure 26:*Actual and Predictions of the open stock price using CNN LSTM*

From Figure 26, we can visualize the predictions of CNN LSTM which are good, capture the trends in data, and fall in 95% prediction intervals. Similar to Bidirectional LSTM, CNN LSTM predictions also have a high difference with actual data when the price drops.

6.6 MULTIVARIATE LSTM ANALYSIS

In all the above-mentioned analyses, we use only the open stock price feature and predict it through the available dates, however in the multivariate analysis we use Open, Close, High, Low, and Adjustable close price data and give them as inputs to the data and predict for the open stock price. This model is useful in capturing other feature trends and predicting accordingly. For this model, we need not perform feature extraction as we are using all features available and the rest process is carried out in a similar to the above analysis i.e., we normalize the data and split it into train and test data sets as explained in chapters 5.2.2 &5.3. Then we initialize hyperparameters as Units=64, Batch size = 32, Dropout = 0.2, Optimizer = 'adam', epochs = 50, linear activation function for dense layer. Early stopping and learning rates would be implemented automatically based on Validation loss. And for this analysis, we will have stacked LSTM layers, but the model's input data would be of 5 features here. Then we fit the model for train data with test data as validation data and evaluate results. Later we tune hyperparameters having Units=[16,32,64], Batch size=[32,64,128], Optimizers=['adam', 'adamax', 'RMSprop', 'SGD'], and epochs = [20,50,100] whereas dropout=0.2. For all the above hyperparameters we get a total of 108 combinations of hyperparameters with the cartesian product as mentioned in 6.2. Then we iterate through each of the above combinations, fit the model and evaluate the results and store them in a list. Later we arrange the results based on Test data MSE in ascending order and take only the top 5 snippets as going below might increase the loss rate.

Table 12: *Hyperparameter combinations and results for Multivariate LSTM*

SL.No	Units	Batch Size	Dropout	Optimizer	Epochs	Train data MSE	Test data MSE
1	64	32	0.2	Adam	100	0.000313	0.000131
2	32	64	0.2	Adam	100	0.000349	0.000132
3	64	128	0.2	Adam	100	0.000344	0.000132
4	64	64	0.2	Adam	100	0.000250	0.000137
5	64	32	0.2	Adamax	50	0.000255	0.000140

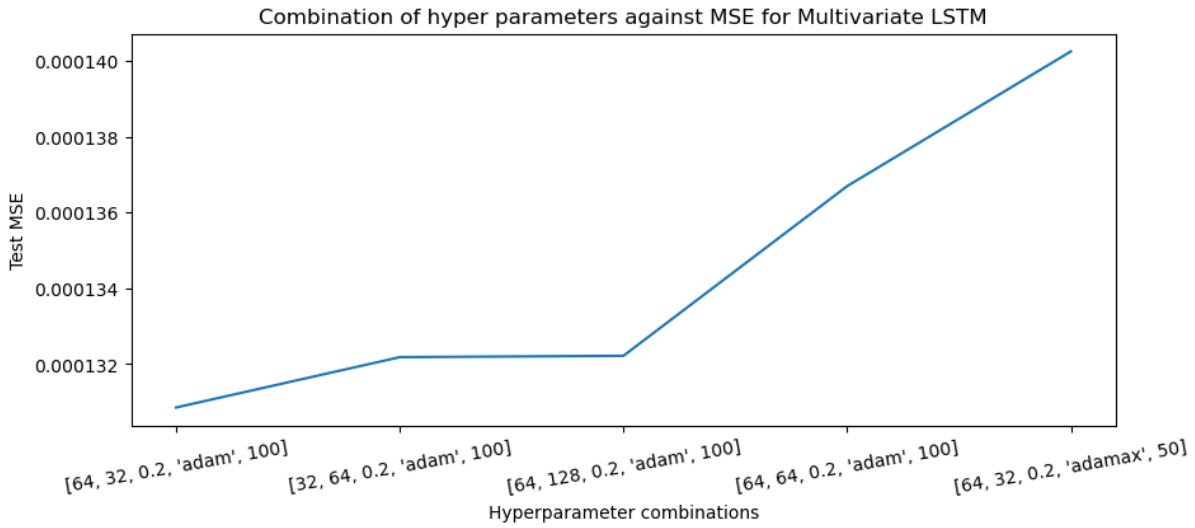


Figure 27: *Hypermater combinations against Test MSE for Multivariate LSTM*

From Table 12 and Figure 27, we see the top 5 combinations of hyperparameter and from this, we assume the 3rd combination i.e., Units = 64, batch size=128, dropout=0.2, Optimizer = 'Adam' and epochs =100 as the best combination because its MSE is neither too low nor too high so this might give optimum model. Now let us compare the results of this model with the initial hyperparameters model and see if there is any improvement in results.

Table 13:*Results of Test set MSE before and after tuning hyperparameters for Multivariate LSTM in a normalized scale*

Results before tuning Test set MSE	**Results after tuning Test set MSE**	**Improvement (%)**
0.000129	0.000132	-3%

From Table 13, we notice there is a decrease in the improvement percentage, so the initial hyperparameter might overfit data because of its less MSE. So, we would be finalizing 3rd combination from table 12 as hyperparameters for the Multivariate LSTM model, then fit the model, evaluate the results and predict for train, test data, and 100 future days. Later we would be de-normalizing the data to get predictions in actual scales, with the help of predictions and actual values we calculate train RMSE and test RMSE in original scales and forecasting errors to get a 95% prediction interval.

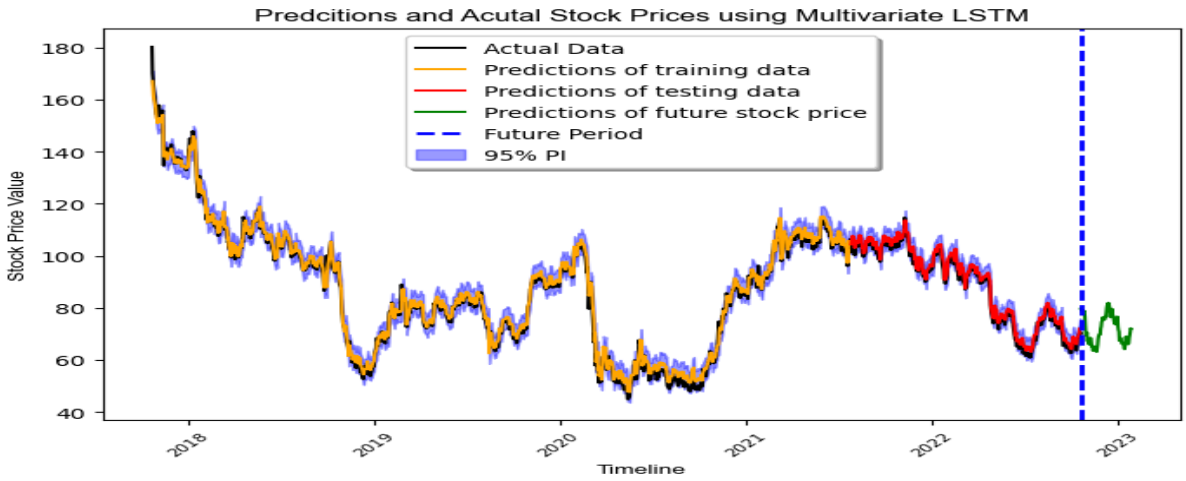


Figure 28: *Actual and Predictions of the open stock price using Multivariate LSTM*

From Figure 28, we can notice that Multivariate LSTM produced predictions that are good, capturing trends in data well and falling in the 95% prediction interval.

7. EXPERIMENT RESULTS

In chapter 6, we discussed all the analysis models and plotted their predictions on the actual data, however, we also predicted for 100 future days for these days we do not have actual data to compare its predictions, however, we can notice the cyclic patterns in these predictions for all models (Fig 20, 22, 24, 26, 28). Table 14 shows the 'first and last 5 days' predictions of the future 100 days, we can also notice that there are patterns.

Table 14: *First and Last 5 days predictions in future 100 days for all models*

Date	Single LSTM	Stacked LSTM	Bidirectional LSTM	CNN LSTM	Multivariate LSTM
2022-10-21	78.136894	78.049835	78.434647	78.491081	79.709145
2022-10-22	78.385887	78.303528	78.673508	78.732986	79.683083
2022-10-23	78.769150	78.694000	79.041344	79.105621	79.051895
2022-10-24	78.022011	77.932777	78.324471	78.379509	79.256821
2022-10-25	76.578224	76.461456	76.941376	76.980125	78.077766
2023-01-24	68.242874	67.965195	69.013161	69.000038	68.494400
2023-01-25	68.638435	68.368233	69.387222	69.374840	69.475616
2023-01-26	69.968391	69.723595	70.646477	70.637909	70.623360
2023-01-27	70.866203	70.638702	71.497940	71.493057	72.183662
2023-01-28	70.714905	70.484482	71.354370	71.348801	71.974731

From Figures 20, 22, 24, 26 & 28 in chapter 6, we notice that all models are predicting well and capturing the trends, however, it seems difficult to come to a final model from these plots as the actual values and predictions are merging almost throughout the data. So, we need to evaluate the model's performance with the help of evaluation measures such as Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R-squared value. Their formulas are given below in equations 2&3.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(Actual_i - Predicted_i)^2}{n}} \rightarrow (equation\ 2)$$

$$MAE = \frac{\sum_{i=1}^{n}|Actual_i - Predicted_i|}{n} \rightarrow (equation\ 3)$$

Where n is the total number of observations.

Table 15: *RMSE, MAE, and R-Squared values of models on Train and Test datasets*

SL.No	Model	Train RMSE	Test RMSE	Train MAE	Test MAE	Train R^2	Test R^2
1	Single LSTM	2.471786	1.911679	1.745843	1.495346	0.98989	0.98338
2	Stacked LSTM	2.408156	1.870703	1.690938	1.459386	0.99040	0.98408
3	Bidirectional LSTM	2.763955	2.014696	2.011442	1.587917	0.98736	0.98154
4	CNN LSTM	2.859503	1.852426	1.997127	1.436413	0.98647	0.98439
5	Multivariate LSTM	2.184506	1.562098	1.548867	1.178798	0.99210	0.98890

From Table 15, we notice RMSE, MAE, and R^2 values of Train and Test datasets for all build models, and in this Multivariate LSTM has the lowest RMSE and MAE for both Train and Test sets and the highest R^2 value for Train and Test sets thereby concluding it the best model for the GE stock price prediction. In 4.4.2 we discussed that Stacked LSTM provides better results than Single LSTM, this can be seen in Table 15. CNN LSTM and Bidirectional LSTM are not good in stock price prediction comparatively as we can see that they have high RMSE, MAE, and low R^2 values.

As discussed in chapter 2, Patel et al [6][7] used methods like SVM, ANN, SVM-ANN, and SVM-RF for predicting stock price and stock price index, however, it was lacking in long-term dependencies by not predicting well after a certain period and failed to get real-time predictions. In our experiment, we conducted the analysis using LSTM which had overcome the limitations and disadvantages of using methods mentioned by Patel et al [6]. And also explained in chapter 3, are the disadvantages of using ANN. No historical data was considered in models built by Nam K and Seong N et al [10], our models were built using historical data from yahoo.finance.com [31]. Rezaei H et al [15] used CNN LSTM whose predictions were good, but they failed to verify the LSTM models using other methods and also not performed hyperparameter tuning to improve the accuracy of the model, we tuned the hyperparameters for each LSTM model built and improved the accuracy of models. Lu et al [16] performed a combined CNN-BiLSTM-AM method for predicting the Shangai stock price prediction, the combined method is high complexity to build and because of this it predicts the data to near accurately making it overfit model and predicting well only on the Shangai data, so we did not build complex models to have the flexibility of predicting other data as well. Chong E et al [8],

failed to get accuracy on the testing set, however, from Table 15, we can notice that accuracy on the testing set is also good in our built models.

8. ASSUMPTIONS / LIMITATIONS

In this analysis, while model building, the hyperparameter combinations which gave optimum MSE i.e., neither too low nor too high are assumed to be the best combination for each model. Even though the reason for choosing such a way is rational, there might be some other way to prove this mathematically as well, which is not performed in these experiments. Only 3 values are chosen for each hyperparameter and 1 value for dropout, this is one of the major limitations of the model, as there is a possibility of having more values for each of them, which could have helped in finding even better accuracy in the model. The model we built is Sequential (), we add the LSTM layer to this Sequential, this does not support AIC (Akaike Information Criterion) function which is used to compare different models and tell which model fits the data well. Assuming that the models built perform best for other company stocks and provide almost similar accuracy based on stock price values. No financial news is considered, so the predictions made for future days are completely based on the input data, this might or might not be accurate because of the market fluctuation. If given more time, we can overcome the limitations on the number of hyperparameters and we can also take financial news into account thereby explaining the reasons for the increase or decrease in the stock price.

9. CONCLUSION

In this report, we obtained historical data from finance.yahoo.com for General Electric Company Stock price from the years October 2017-October 2022. This data all Key Performance Indicators (KPIs) of the stock market such as the open, close, high, low, and adjustable close of the stock price on a respective day. Before performing the analysis, I read a few journals and research papers used for stock price prediction, these papers include different machine learning techniques, however, there were some limitations for most of them as mentioned in Table 1. From these papers, I concluded deep learning technique LSTM and types of LSTM are good for predicting stock price data. Then I discussed the methodology of Deep Learning, Recurrent Neural Networks, Long-Short Term Memory (LSTM), Single Layer LSTM, Stacked LSTM, Bidirectional LSTM, Convolutional Neural Network LSTM, Multivariate LSTM and Hyperparameters of the LSTM.

Initially, after retrieving the data, we performed initial data analysis, from this we understood we need to perform data pre-processing such as feature extraction and normalization i.e., as we are predicting for open stock price, we extracted only open price values from the data for all univariate analysis (Single layer LSTM, Stacked LSTM, Bidirectional LSTM, and CNN LSTM), feature extraction is not required for Multivariate LSTM models as these are capable of taking multiple features at a time and produce predictions for a single feature, then normalized the data to reduce the loss and increase the accuracy of the model and this also helps in reducing the unwanted bias in the data. Afterward, we split the data into train and test datasets, and with the help of these datasets, we trained and tested the models. Before building

the models, we set initial hyperparameters as Units=64, Batch size =32, Dropout = 0.2, Optimizer = 'adam', and epochs=50 for all the built models and evaluated results. As there is a possibility of overfitting or underfitting the model with the above hyperparameters we tuned them by setting Units=[16,32,64], Batch size=[32,64,128], Optimizers=['adam', 'adamax', 'SGD', 'RMSprop'], epochs=[20,50,100] and dropout = 0.2. After tuning each model had a different set of hyperparameters as best shown in Table 16, these sets helped in finding the optimum model by overcoming underfit for Single Layer LSTM, Stacked LSTM, Bidirectional LSTM, and CNN LSTM, and for Multivariate LSTM they had overcome the overfitting of the model.

Table 16: *Models with their best hyperparameter combinations*

SL.No	Model	Units	Batch size	Dropout	Optimizer	Epochs
1	Single LSTM	64	64	0.2	RMSprop	100
2	Stacked LSTM	64	128	0.2	Adam	50
3	Bidirectional LSTM	64	64	0.2	RMSprop	50
4	CNN LSTM	64	64	0.2	Adam	20
5	Multivariate LSTM	64	128	0.2	Adam	100

Later I built models with the best hyperparameter combinations for each model and predicted for train, test, and 100 future days values. Then de-normalized the data and predictions to actual scales, with the help of these 95% Prediction intervals, RMSE, MAE, and R^2 values are obtained. Afterward, we plotted actual values and predictions for all models, however, it was difficult to the find best model because of near-accurate predictions, so with the help of evaluation measures such as RMSE, MAE, and R^2 value, I compared the performance of models. Later, I discussed the assumptions made while model building and their limitations.

Finally, I conclude that Multivariate LSTM is the best, as it takes trends of other features as well in predicting the data and also has good evaluation measure values i.e., the lowest RMSE and MAE, and the highest R^2 values. Even in general open stock price is completely dependent on other features, Multivariate LSTM takes all those features also into account in model building. And, if predictions are to be performed for only one feature that is not affected by other features, Stacked LSTM would be good as it provides good evaluation measures after Multivariate LSTM.

A range of factors, such as government policies, business operations, interest rates, and so on, are known to influence stock prices. News about any of these elements has an impact on stock prices. Natural disasters and other unforeseen occurrences will have a substantial impact on any good machine learning or deep learning model. As a result, a hybrid technique that takes into account all prospective variables such as news, attitudes, and technical indicators can be built, resulting in a more robust and accurate prediction system.

REFERENCES

[1] Raghav Nandakumar, Uttamraj K R, Vishal R, Y V Lokeswari. “Stock Price Prediction using Long-short term memory”. Available at: https://www.irjet.net/archives/V5/i3/IRJET-V5I3788.pdf

[2] Soulas, Eleftherios, and Dennis Shasha. “Online machine learning algorithms for currency exchange prediction.” Computer Science Department in New York University, Tech. Rep 31 (2013).

[3] Recht, Benjamin, et al. “Hogwild: A lock-free approach to parallelizing stochastic gradient descent.” Advances in neural information processing systems. 2011.

[4] S. Chakraborty. “Capturing financial markets to apply deep reinforcement learning”, 2019.

[5] T. Moyaert and M. Petitjean. “The performance of popular stochastic volatility option pricing models during the subprime crisis. Applied Financial Economics”. Pg.no: 21(14), 2011.

[6] Patel, J., Shah, S., Thakkar, P., & Kotecha, K. (2015). “Predicting stock and stock price index movement using trend deterministic data preparation and machine learning techniques”. Expert systems with applications, 42(1), 259-268.

[7] Patel, J., Shah, S., Thakkar, P., & Kotecha, K. (2015). “Predicting stock market index using the fusion of machine learning techniques”. Expert Systems with Applications, 42(4), 2162-2172.

[8] Chen, Y., & Hao, Y. (2017). “A feature weighted support vector machine and K-nearest neighbor algorithm for stock market indices prediction”. Expert Systems with Applications, 80, 340-355.

[9] Chong, E., Han, C., & Park, F. C. (2017). “Deep learning networks for stock market analysis and prediction: Methodology, data representations, and case studies”. Expert Systems with Applications, 83, 187-205.

[10] Nam, K., & Seong, N. (2019). “Financial news-based stock movement prediction using causality analysis of influence in the Korean stock market”. Decision Support Systems, 117, 100-112.

[11] Yang, F., Chen, Z., Li, J., & Tang, L. (2019). A novel hybrid stock selection method with stock prediction. Applied Soft Computing, 80, 820-831.

[12] Long, J., Chen, Z., He, W., Wu, T., & Ren, J. (2020). “An integrated framework of deep learning and knowledge graph for prediction of stock price trend: An application in Chinese stock exchange market”. Applied Soft Computing, 91, 106-205.

[13] Yan, B., & Aasma, M. (2020). “A novel deep learning framework: Prediction and analysis of financial time series using CEEMD and LSTM”. Expert systems with applications, 159, 113609.

[14] Jing, N., Wu, Z., & Wang, H. (2021). “A hybrid model integrating deep learning with investor sentiment analysis for stock price prediction”. Expert Systems with Applications, 178, 115019.

[15] Rezaei, H., Faaljou, H., & Mansourfar, G. (2021). “Stock price prediction using deep learning and frequency decomposition”. Expert Systems with Applications, 169, 114332.

[16] Lu, W., Li, J., Wang, J., & Qin, L. (2021). "A CNN-BiLSTM-AM method for stock price prediction". Neural Computing and Applications, 33(10), 4741- 4753.

[17] Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training recurrent neural networks." International Conference on Machine Learning. 2013.

[18] Soulas, Eleftherios, and Dennis Shasha. "Online machine learning algorithms for currency exchange prediction."Computer Science Department in New York University, Tech. Rep 31 (2013).

[19] Altinbas, H., Biskin, O. T. "Selecting macroeconomic influencers on stock markets by using feature selection algorithms". Procedia Economics and Finance, 30, 22-29. (2015).

[20] The Analysis and Prediction of Stock Price; 2013 IEEE International Conference on Granular Computing (GrC).

[21] Sima Siami-Namini, Neda Tavakoli, Akbar Siami Namin. "A Comparison of ARIMA and LSTM in Forecasting Time Series" 2018 17th IEEE International Conference on Machine Learning and Applications. Available at: https://par.nsf.gov/servlets/purl/10186768

[22] Vaibhav Kumar. "Comparing ARIMA Model and LSTM RNN Model in Time-Series Forecasting". Available at: https://analyticsindiamag.com/comparing-arima-model-and-lstm-rnn-model-in-time-series-forecasting/

[23] Y. Yu, X. Si, C. Hu, and J. Zhang. "A review of recurrent neural networks: Lstm cells and network architectures. Neural Computation", 31(7):1235–1270,2019.

[24] Janki Patel, Prof. Miral Patel, Prof. Mittal Darji. "Stock Price Prediction Using RNN". Available at: https://www.jetir.org/papers/JETIRK006164.pdf

[25] Y. Bengio, P. Frasconi, and P. Simard. "The problem of learning long-term dependencies in recurrent networks". In IEEE International Conference on Neural Networks, pages 1183–1188 vol.3, 1993.

[26] S. Hochreiter and J. Schmidhuber. "Long short-term memory. Neural Comput"., 9(8):1735–1780, Nov. 1997.

[27] Kate brush and Ed Burns. "What is deep learning and how it works?". Available online at:https://www.techtarget.com/searchenterpriseai/definition/deep-learning-deep-neural-network

[28] A. Graves, N. Jaitly, and A. r. Mohamed. "Hybrid speech recognition with deep bidirectional lstm," in Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on. IEEE, pp. 273–278. 2013.

[29] M. Schuster and K. Paliwal. Bidirectional recurrent neural networks IEEE Transactions on Signal Processing, 45, pp. 2673-2681. 1997.

[30] "Model Check points Tensorflow" Available online at: https://www.tensorflow.org/swift/guide/checkpoints

[31] "General Electric Company Stock price data" Available at: https://finance.yahoo.com/

[32] Shreesti Singh. "Pooling layers for CNN". Available at: https://www.linkedin.com/pulse/pooling-layers-convolutional-neural-networks-sreeshti-singh/