Applications of Neural Networks with Long Short-Term Memory

High Integrity System
Learning From Data
Anik Saha
1325658
anik.saha@stud.fra-uas.de

***Abstract*— Learning to store information over long time intervals using recurrent backpropagation requires a very lengthy time, owing to insufficient, fading error backflow. To circumvent this, the researcher devised a Long Short-Term Memory (LSTM) network. In recent years, these networks have emerged as cutting-edge models for a wide range of challenges where sequence of data need to be required. In this work, we explore the fundamental structure of LSTM and provide an example of LSTM implementation for forecasting time series data.**

Keywords—RNN, LSTM, Time Series, Activation Function

I. INTRODUCTION

Neural networks are one of the most important inventions in the programming world. In the traditional method, the computer is told what to do. Furthermore, large problems are broken down into smaller subproblems, and tasks are properly stated so that the computer can understand and complete the work. A neural network, on the other hand, solves problems for itself. It learns from facts collected through observation and develops its own solution to the problem. Except for a few specialized issues, according to Michael Nielsen, people did not know how to train neural networks in such a way that they could beat more conventional approaches until 2006. However, things changed in 2006, when techniques for learning so-called deep neural networks were discovered. Deep learning is the name given to these techniques[1]. Deep learning, which is inspired by conventional neural networks, outshines its predecessors drastically. It develops multi-layered learning models by combining graph technologies with transformations among neurons. Currently lots of deep learning approaches have been shown good performance in a variety of applications, which includes Natural Language Processing (NLP), visual data processing, voice and audio processing, and a variety of other well-known applications [2]-[4].

In this paper [105], Jürgen Schmidhuber defines a neural network (NN) as a collection of several simple, connected processors called neurons, each of which produces a series of real-valued activations. Sensors that monitor the environment activate input neurons, whereas weighted connections between previously active neurons stimulate new neurons [5]. Neural networks can be categorized depending on the presence of connections between neurons within the same layer: Feed Forward Neural Networks (FFNNs) and Recurrent Neural Networks (RNNs) (RNNs). In FFNNs, there is no communication between neurons in the same layer, and all neurons cannot be linked across layers, thus information only travels in one way, from the input layer to the output layer, through any hidden layers (if any). By calculating the output of the current instant from the input of the current moment and the hidden state of the previous moment, RNNs allow neurons within the same hidden layer to be linked. As a result, RNNs incorporate historical input data in the internal state of the network and therefore it is capable of translating all previous input data to the final output. In theory, RNNs are better than FFNNs in dealing with long-range dependencies [6].

LSTMs are a special form of RNN that addresses some of the drawbacks of recurrent networks. They were first presented by Hochreiter & Schmidhuber in 1997 [7], and they have since been developed and popularized by a slew of other researchers. They are now widely used and perform exceptionally well on a wide range of situations. LSTMs are specifically intended to prevent the problem of long-term reliance. Remembering information for lengthy periods of time is practically their default behavior, not something they struggle to learn [8].

Forecasting may be described as the study of historical data in order to predict future events or occurrences. Business and industry, economics, environmental science, and finance, to name a few, are among the fields covered. Many forecasting difficulties necessitate the study of time. A time series data set is a collection of observations for a specific variable in a chronological order. It might be either univariate or multivariate in nature. Multivariate data includes information about many attributes at various periods in time, whereas univariate data contains information about only one property. The finding of patterns, trends, and periods or cycles in data is aided by time series data analysis. Knowing the optimistic or negative sentiment of the stock market early on can help investing money properly. Pattern analysis may also be used to identify the best-performing firms over a period of time. As a result, time series analysis and forecasting have become important research topics [9].

We begin by discussing RNN and its limitations in this paper. Then we'll go through an overview of LSTM and some of its applications. Following that, we explore several types of activation functions in neural networks and pros and cons of those activation functions. Then, we demonstrate how to use LSTMs to forecast stock open prices for a specific company. Finally, we explore some of the issues that LSTMs have and draw some conclusions.

II. RNN AND ITS LIMITATIONS

In this paper [10], Siam Siami Namin and Akbar Siami Namin noted that RNN are a special type of neural network. The objective of it is to anticipate the next step in a sequence of observations based on the previous stages in the sequence. The aim of RNNs, in reality, is to predict future outcomes using sequential observations and learning from previous phases. As a result, data from previous stages must be recalled for projecting future steps [10]. This characteristic is critical in many situations where embedded

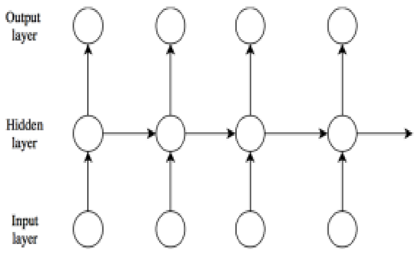


Fig. 1 RNN Structure [2]

structure in the data sequence provides important information. For example, knowing the context is required to comprehend a word in a sentence. As a result, an RNN may be thought of as a collection of short-term memory units, with input layer x, hidden (state) layers, and output layer y.

In Fig.1, a typical unfolded RNN diagram for an input sequence is shown. In this paper [47], Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio provide three deep RNN approaches: deep "Input-to-Hidden," "Hidden-to-Output," and "Hidden-to-Hidden." Based on these three techniques, a deep RNN is presented, which not only benefits from a deeper RNN but also reduces the complexity of learning in deep networks [2]. However, RNN has several drawbacks, which are detailed below:

Vanishing Gradient and exploding gradient Problem: Recurrent networks, like other types of neural networks, are old-fashioned. When it came to recurrent performance, the Vanishing Gradient proved to be a major roadblock. We can express gradient as a straight line where the change in all weight(x-direction) with respect to the change of error(y-direction) [11]. As a result, weight must be adjusted to reduce the error of a network; otherwise, our neural network will struggle to learn and will be unable to produce proper predictions.

The updated weight in a specific layer of a RNN can be determined by multiplying the learning rate, the error from the previous layer, and the input of the current layer. The product of all errors that occur on previous layers is the error of a particular layer. The modest values of the activation function's derivative are multiplied many times. The Long-Term component approaches norm "0" exponentially quickly, which occur vanishing the gradient from the network. When a network's gradient is nearly vanishing, it's difficult to train the network accurately. Furthermore, when the gradient is propagated across the network, it has the potential to rise exponentially from layer to layer. Exploding gradients is the name given to this situation [12]. The LSTM was created by Hochreiter and Schmidhuber to solve these gradient problems [7]. Unlike RNN, LSTM has three gates: an input gate, a forget gate, and an output gate. As a result, it has control over what must be kept and what must be forgotten. That is why LSTM can hold information from the past, whereas RNN cannot [13].

III. The Concept of Long Short-Term Memory

As previously mentioned, LSTM is a kind of RNN architecture that is recognized for memorizing data across arbitrary intervals and is well-suited to classify, analyze, and forecast time series with unknown time delays. One of the benefits of LSTM is that it is unaffected by gap length. Before emitting/sequencing, RNNs rely on the hidden state. All RNNs are made up of a series of RNN modules, each of which has a basic structure such as a single tanh layer. LSTM, on the other hand, has a chain-like structure, but it contains four layers to regulate data flow [14].

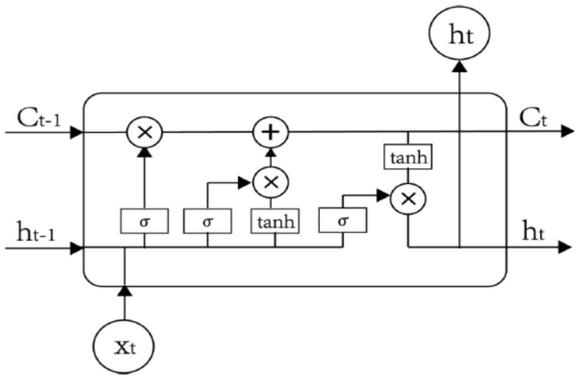


Fig. 2 Basic Structure of LSTM [15]

The LSTM's fundamental design consists of cell states and several gates. The long-term memory is usually referred to as the cell state [14]. Consider cell state as a highway that only carries relevant data down the sequence chain. The cell state may be thought of as the LSTMs' "Memory." The cell state transfers critical information from one timestep to the next, decreasing the effect of short-term memory. Information is added or removed from the cell state via gates [46]. The LSTM model filters input through the gate structure to maintain and update the state of memory cells. Its gate structure consists of input, forgotten, and output gates. There are three sigmoid layers and one tanh layer in each memory cell. In Fig. 2 the structure of LSTM memory cells is shown [15].

Gates are a mechanism to selectively allow information to pass through. A sigmoid neural net layer plus a pointwise multiplication operation makes them up. The sigmoid layer produces values ranging from zero to one, indicating how much of each component should be allowed to pass. A value of zero indicates that "nothing should be allowed through," whereas a value of one indicates that "everything should be allowed through!" [14]. Fig. 3 depicts the structure of basic gates.

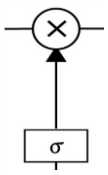


Fig. 3 Basic Structure of Gate [14]

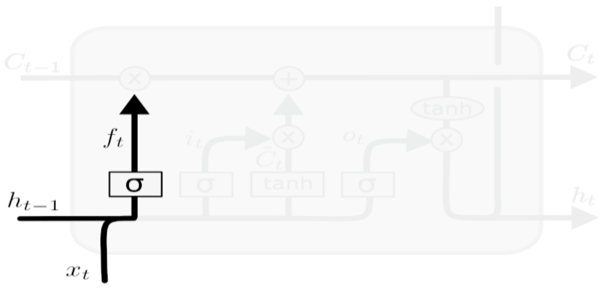


Fig. 4 Forget Gate [14]

In the initial stage of the LSTM unit, the Forgotten gate decides which cell state information is erased from the model. The memory cell, as shown in Fig. 1, takes the previous moment's output h_{t-1} and the current moment's external information x_t as inputs and combines them into a long vector $[h_{t-1}, x_t]$ through σ transformation to become

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

Where W_f and b_f are, respectively, the weight matrix and bias of the forgotten gate, and σ is the sigmoid function. The forgotten gate's major duty is to keep track of how much of the previous time's cell state C_{t-1} is reserved for the current time's cell state C_t Based on h_{t-1} and x_t, the gate will output a value between 0 and 1 [15]. The structure of the forget gate is represented in Fig. 4.

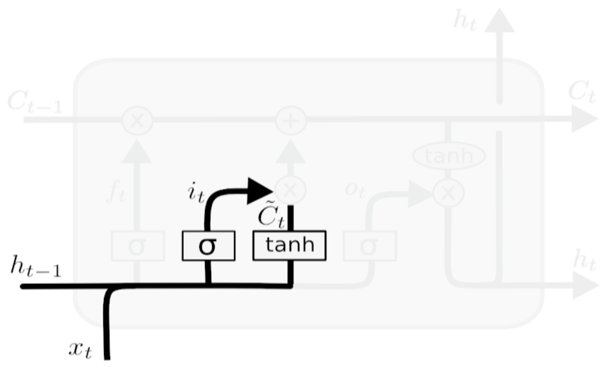


Fig. 5 Input Gate [14]

The following step is to determine what additional data will be saved in the cell state. This is accomplished through the use of an input gate. It defines how much of the current time network input x_t is reserved for the cell state C_t, preventing irrelevant data from reaching memory cells. It has two functions. The first step is to determine the state of the cell that needs to be updated; the sigmoid layer selects the value to be updated, as indicated in Eq (2) [15]. Fig. 5 shows the functionality of input gate demonstrated. The second process is to update the information to the cell's current state. As demonstrated in Eq (3), the tanh layer is used to control how much new information is stored to a new candidate vector.

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

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

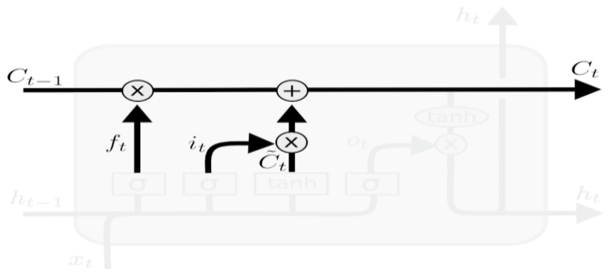


Fig. 6 New Cell State [14]

It is now time to update the previous cell state, C_{t-1}, to the new cell state, C_t. The preceding phases have already determined what should be done; all that remains is for us to carry it out. The old state by f_t is multiplied, forgetting the things that is decided to forget earlier. Then $i_t * C_t$ is added. This is the updated set of candidate values, scaled by the amount by which it is decided to change each state value [15] in Eq 4. The Fig. 6 demonstrates the whole functionality.

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C_t} \quad (4)$$

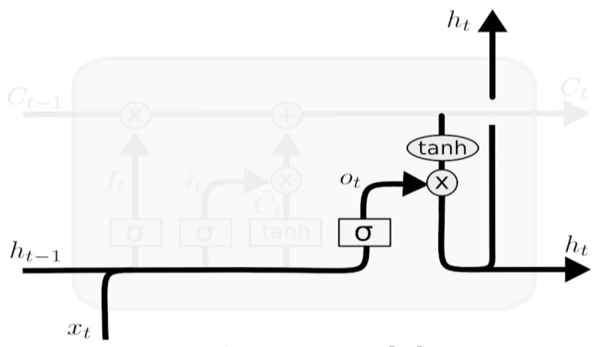


Fig. 7 Output Gate [14]

Finally, it is time to decide what is going to output as showed in Fig.7[15]. This result will be dependent on the state of our cells. It also regulates how much of the current cell state is thrown away. The output information is first decided by a sigmoid layer, which changes the output value between zero and one once again, and then the cell state is processed by tanh and multiplied by the sigmoid layer's output to create the final output portion [15] shown in Eq (5):

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

This tanh converts a minus one to a plus one number. In Eq(6), the cell's ultimate output value is defined as:

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

To summarize, the forget gate determines which information must be kept or discarded, the input gate decides which information should be added to the cell state, and the output gate defines the next hidden state [16].

IV. The Applications of LSTM

LSTM is currently being utilized to tackle sequence problems in a variety of applications, including robot control, speech recognition, and time series forecasting. It can classify, analyze, and anticipate time series and reflects major occurrences by learning from historical data. The findings show the potential of using an LSTM network to anticipate retail sales [17]. Language modeling with spontaneous phone calls is one application of the LSTM network. Experiments have shown that this improves perplexity and word error rate (WER) on the recognition system [18]. Researchers also found good result in Robot Command Interaction Modeling by using multi-layer LSTM [19].

By applying a long short-term memory (LSTM) deep learning method, a novel trial to reproduce soil stress–strain behavior is being conducted. LSTM is a method for predicting future events based on time sequence data, and it can be used to the stress history of soil behavior. The approach continues with data preparation, architecture determination, and optimization. LSTM networks have recently been applied in civil engineering to model soil moisture and the hydro-mechanical coupling effect of porous media, among other things. However, few research have used the LSTM deep learning method to explore the stress–strain behavior of soils [20].

Another application of LSTM is the prediction of water usage as a foundation for reliable water distribution plant operation planning. The outcomes of optimal pump control based on water demand prediction using the analyzed LSTMs, as well as associated work in terms of time series prediction, setting up workable LSTM models for use cases, and other research concepts, all contribute to better operational planning [21]. In text generation, the next character or word in the sequence must be predicted or a huge amount of preceding data must be memorized. Text data is commonly considered as a set of data. The LSTM is favored in this case because it predicts data in the sequence as the deep learning model [22].

Individual words' phons (phonemes) can be identified with greater accuracy and efficiency using LSTM [23]. As a result, for end-to-end ASR models, it achieves better outcomes in terms of word error rate (WER) [24]. There is sufficient evidence that deep neural networks can model speeches efficiently in a variety of languages when it comes to speech recognition. For image captioning, an LSTM-based model is employed to predict the sequences, called caption, from feature vectors produced from the VGG network [25]. For instance, consider employing the deep CNN architecture to extract features from an image, which are then fed into the LSTM architecture to generate the caption.

The LSTM network uses a gating mechanism to recognize and encode long-term patterns. It aids in the retention of knowledge over a lengthy period of time, as in music and text generation. Clearly, the LSTM network can accommodate a large number of different output predictions [26]. Chang, Yang Zhang, and Wenbo Chen employed Adam optimized LSTM to develop forecasting model that improves the accuracy of New South Wales power price prediction. There are a plethora of alternative uses for LSTMs. For instances, Sentiment Analysis (Input is text and output is positive or negative) [29], Video Activity Recognition (input is video and output is type of activity) [28], Handwriting recognition [27] and so on. Besides, LSTMs are used in a variety of medical applications. DNA sequence analysis is one example. Financial applications such as stock price prediction [30] use it as well.

V. Activition Function

A biological neuron receives electrical messages from other neurons of various weights. The neuron enters the excited state to provide a response when the synthetic value of all electrical signals is large enough to stimulate it. Otherwise, it will remain inactive [31]. To determine whether the neuron should be turned on or off, the activation function is applied to the weighted sum of inputs and bias. The output of each neuron is transferred to the next layer, and the type of activation function used is determined by the intended output.

A. *Types of activation functions*

There are different types of activation function. Below different activation and their pros and cons is discussed.

1) *Sigmoid function*

One of the most popular types of activation functions is the sigmoid function. The sigmoid function is a continuous function, which implies it can be differentiated everywhere. The derivatives of a sigmoid function are simple to compute. As a result, the sigmoid function was widely employed in shallow neural networks. Furthermore, due to its value distribution, the sigmoid function is typically used in the output level of a neural network. Because it only achieves zero gradient in the limit, it is soft saturate [32]. The soft saturation makes training a deep neural network difficult. More specifically, when optimizing the loss function, the derivatives of the sigmoid function, which are used to update the weights and bias, will fall to zero in the saturation area, resulting in less contributions from the first several layers in knowledge learning from training samples. In fact, this is known as a vanishing gradient [33].

2) *Hyperbolic Tangent function*

The hyperbolic tangent function is the ratio of the sine and cosine functions. The hyperbolic tangent function has an output range of -1 to 1. It is also a continuous and monotonic function that can be differentiated everywhere. It is symmetric about the origin, and the outputs, presumably the inputs of the next layer, are more likely to be near to zero on average. As a result, hyperbolic tangent functions are chosen over sigmoid functions. Furthermore, neural networks with hyperbolic tangent activation functions converge quicker than neural networks with sigmoid activation functions [34]. Furthermore, neural networks with hyperbolic tangent activation functions outperform those with sigmoid activation functions in terms of classification error [33]. The calculation of the derivatives of hyperbolic tangent functions, on the other hand, is more difficult than that of the sigmoid function. It also has the same soft saturation as the sigmoid function, which has the vanishing gradient problem [33].

3) *ReLU function*

Cortical neurons are rarely at their maximum saturation regime, according to neuroscience studies, and their activation function can be approximated by a rectifier [35]. More specifically, only one percent to four percent of neurons in the brain can be engaged at the same time. However, in neural networks with sigmoid or hyperbolic tangent activation functions, about half of the neuron units are active at the same time, which contradicts neuroscience studies. Furthermore, activating more neuron units will make training a deep neural network more difficult [36].

The ReLU function and its enhancements are currently the most widely used activation functions in deep neural networks [33], [37]-[39]. Although the idea of initializing each layer by unsupervised learning is said to solve the major challenge of training deep networks, the use of ReLU activation functions is also seen as a breakthrough in directly supervised deep network training. The activation function of the ReLU is non-saturated, and its derivative function is constant when the input is greater than 0. As a result, the vanishing gradient problem can be solved. The ReLU function, in particular, provides the following advantages [33], [37]:

- As there is no need to compute the exponential functions in activations, neural network computations with ReLU functions are less expensive than sigmoid and hyperbolic tangent activation functions.
- In terms of training time using gradient descent, neural networks with ReLU activation functions converge considerably faster than those with saturating activation functions.
- The ReLU function makes it simple for a network to obtain sparse representation. More specifically, when the input $x<0$, the output is 0, which provides sparsity in neuron unit activation and enhances data learning efficiency. When the input $x \geqslant 0$, the data's features can be preserved to a significant extent.
- The derivatives of the ReLU function are kept constant at 1, preventing entrapment in local optimization and resolving the vanishing gradient effect seen in sigmoid and hyperbolic tangent activation functions.
- Deep neural networks using ReLU activation functions can achieve their greatest performance on solely supervised tasks with large, labeled datasets without requiring any unsupervised pre-training [31].

Considering all the pros and cons, it is decided to use ReLU as our activation function in our current implementation.

VI. Implementation of Application Using LSTM

This paper attempts to explain how to use LSTM to construct univariate time series forecasting. Our code is highly influenced by Krish Naik's Stock-Market-Forecasting code [40].

A. *Environment*

This system is built on Miniconda and Python (3.7.11). Keras(2.2.4-tf) and TensorFlow(2.1.0) are employed as deep learning platforms. Jupyter Notebook serves as an IDE.

B. *Data Collection*

Yahoo Finance is used to gather Google stock price data. The information covers the period from August 31, 2011, through August 30, 2021. There are a total of 2517 rows. The five columns in the row are Date, Open, High, Low, Close, Adj Close, and Volume.

Table 1 Partial Sock data table

	Date	Open	High	Low	Close	Adj close	volume
0	2011-08-31	271.352	272.130	266.999	269.469	269.469	5406790
1	2011-09-01	269.365	270.899	264.618	265.255	265.255	4849108
2	2011-09-02	261.255	262.974	259.392	261.440	261.440	4820401
3	2011-09-06	254.446	260.403	254.296	260.115	260.115	5451959

The opening price (Open) of a company is the initial transaction price per share when the market begins on a trading day. In univariate time series forecasting, there is only one value used. As a result, the variable Open is used as a predictor variable. The plot diagram of Open variable is showed in Fig. 8. The value of Open is utilized in the Y axis, while the number of days is used in the X axis.

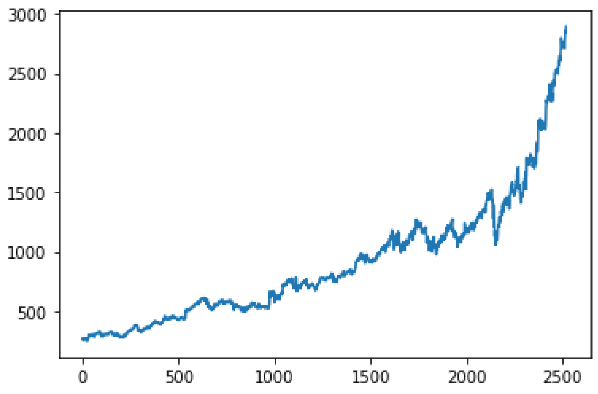
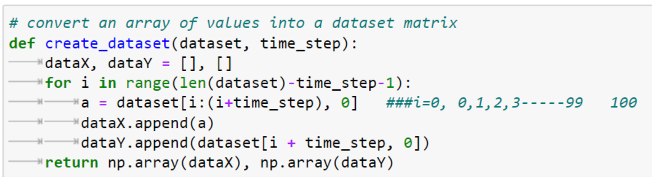


Fig. 8 The Plot Diagram of Open Variable

C. *Data Preprocessing*

Unscaled data might cause a learning process slow and unreliable. We scaled our Open variable. MinMaxScaler from sklearn is used for this. The MinMaxScaler transforms features by scaling them to a specific range [41]. In this program, the range is from 0 to 1. The dataset is then divided into training and test groups. In training set, 65 percent data are employed. The size of training and test sets are 1635 and 881 respectively.

The Open variable dataset must then be turned into a time series data set for our time series. Assume the array is [t1 t2 t3 t4 t5 t6], and two-time step will be used as input. Then output will be t3 for the time step of t1 and t2. The output for t2 and t3 input will be t4. As a result, the time step sets are {(t1, t2), (t2, t3), (t3, t4), (t4, t5)}, while the output set are {t3, t4, t5, t6}. For converting our dataset in time series, create dataset method is used which is shown in Fig. 9.



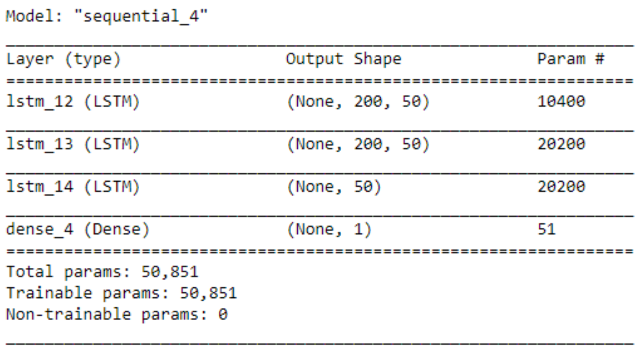
```
# convert an array of values into a dataset matrix
def create_dataset(dataset, time_step):
	dataX, dataY = [], []
	for i in range(len(dataset)-time_step-1):
		a = dataset[i:(i+time_step), 0]   ###i=0, 0,1,2,3-----99   100
		dataX.append(a)
		dataY.append(dataset[i + time_step, 0])
	return np.array(dataX), np.array(dataY)
```

Fig. 9 Create Dataset Method

After creating dataset in time series manner, dataset need to be reshaped in three dimensional array [samples, time steps, features] which is required for LSTM.

D. *Neural Network Architecture*

Three LSTM layers and one dense layer is used to create model. ReLU is used as an activation function in the first and second LSTM layers. Fig. 10 shows deep neural model of our application. For compiling the network, loss function and optimizer need to be set loss. As a result, mean squared error set as loss function and Adam is set as optimizer. Then model fit method is called to train the dataset.



```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_12 (LSTM)               (None, 200, 50)           10400
_________________________________________________________________
lstm_13 (LSTM)               (None, 200, 50)           20200
_________________________________________________________________
lstm_14 (LSTM)               (None, 50)                20200
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 51
=================================================================
Total params: 50,851
Trainable params: 50,851
Non-trainable params: 0
_________________________________________________________________
```

Fig. 10 Deep Neural Network Model

E. *Results*

To evaluate program, the Root Mean Squared Error is used as performance metrics. The training dataset has RMSE value of 671.2217, while the test dataset has a value of 1625.2230717.

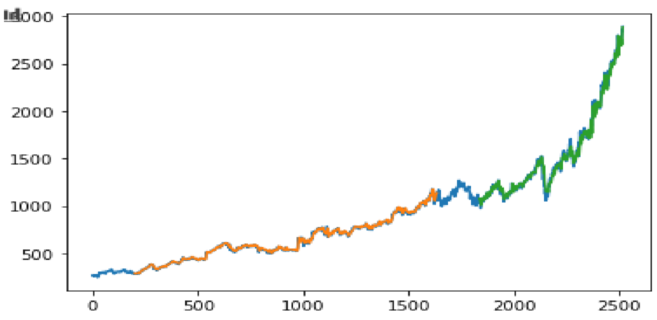
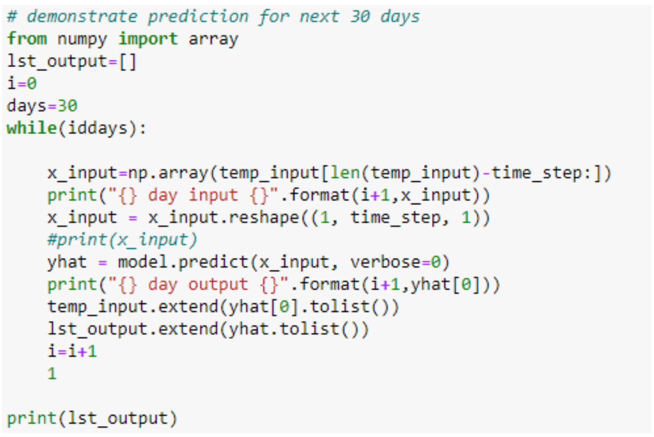


Fig. 11 The Plot Diagram of Predicted Values for Training and Test Data Set

For the Training and Test datasets, the Fig. 11 displays how it predicted the value of the Open variable. The orange color in the graph represents prediction for training dataset, whereas the green color represents test dataset prediction. The actual value for Open pricing is represented by the blue color. We can see from this graph that the forecasted value is very close to the real value of Open.

F. *Forcasting future value*

It is now shown how to use this model to predict future value. To begin, we construct a list of the last 200 days. Then, using the model predict method, we retrieve the predicted value for the 201st day. Then we add a new value to our list while removing the first. This process is carried out until we get all the values next 30 days. This code can be found in Fig. 12.



```
# demonstrate prediction for next 30 days
from numpy import array
lst_output=[]
i=0
days=30
while(iddays):

    x_input=np.array(temp_input[len(temp_input)-time_step:])
    print("{} day input {}".format(i+1,x_input))
    x_input = x_input.reshape((1, time_step, 1))
    #print(x_input)
    yhat = model.predict(x_input, verbose=0)
    print("{} day output {}".format(i+1,yhat[0]))
    temp_input.extend(yhat[0].tolist())
    lst_output.extend(yhat.tolist())
    i=i+1
    1

print(lst_output)
```

Fig. 12 Code of Forecasting Value

Fig. 13 depicts a diagram showing the new forecasted value. The orange color represents the forecasted value for the next 30 days, while the blue color represents the actual value for the prior days.

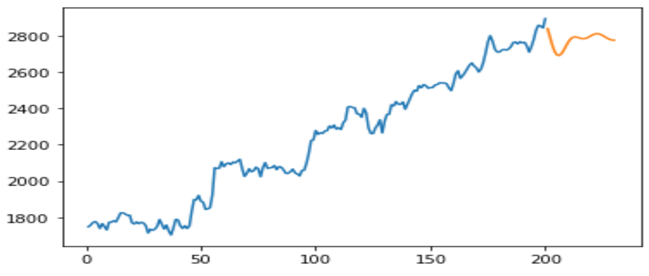


Fig. 13 The Plot Diagram of Forecasting Value

G. *Challenges*

The RMSE difference between the training and test data sets is large in this implementation. Various hyper tuning techniques are utilized to improve this result. The first is changing units in LSTM layers. ReLU activation is also replaced by sigmoid and hyperbolic tangent to improve the results. It is also possible to experiment with different time step numbers as input, such as 100, 150, and 200. At the very least, the generated outcome with these hyper parameters is superior to those obtained previously.

VII. Drawbacks of LSTM

Eugenio Culurciello discussed memory bandwidth problem of LSTMs in his blog. The processing unit(s) require the same amount of memory bandwidth as the number of operations/s they can perform, making full utilization impossible! External bandwidth will never be enough [42]. David Vilares and Carlos Gomez also faced memory problem at the time of training model [43]. Leo Dirac also discusses some the problem of LSTM in his talk. According to him, LSTM need very long gradient path, which create very deep network for long document and that is very difficult to train [44]. Using internal fast caches with high bandwidth can help to alleviate the situation slightly [42]. Considering this situation, researchers developed a new deep learning model called Transformer that outperforms LSTM in terms of stability and training time [45]

VIII. Conclusion

This article discusses how LSTM overcomes RNN gradient issues and the basic framework of LSTM. And an example of LSTM implementation using TensorFlow is demonstrated in this paper. LSTM is very useful for solving problems that require a sequence of data. However, LSTM requires more memory to train and run the model. The future work includes difference of LSTM and Transformer in terms of performance for predicting time series data.

References

[1] M. A. Nielsen, "Neural networks and deep learning," p. 3, 2015.

[2] S. Pouyanfar et al., "A survey on deep learning: Algorithms, techniques, and applications," ACM Comput. Surv., vol. 51, no. 5, pp. 2–5, 2019.

[3] Y. Yan, M. Chen, S. Sadiq, and M.-L. Shyu, "Efficient imbalanced multimedia concept retrieval by deep learning on spark clusters," Int. J. Multimed. Data Eng. Manag., vol. 8, no. 1, pp. 1–20, 2017.

[4] Y. Yan, M. Chen, M.-L. Shyu, and S.-C. Chen, "Deep learning for imbalanced multimedia data classification," in 2015 IEEE International Symposium on Multimedia (ISM), 2015.

[5] J. Schmidhuber, "Deep learning in neural networks: an overview," Neural Netw., vol. 61, p. 86, 2015.Shih C H, Yan B C, Liu S H, et al. Investigating Siamese LSTM networks for text categorization[C]//Asia-pacific Signal & Information Processing Association Summit & Conference. 2017

[6] Y. Hua, Z. Zhao, R. Li, X. Chen, Z. Liu, and H. Zhang, "Deep learning with long short-term memory for time series prediction," IEEE Commun. Mag., vol. 57, no. 6, pp. 114–119, 2019.

[7] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural Comput., vol. 9, no. 8, pp. 1735–1780, 1997.

[8] K . Khalil, B. Dey, A. Kumar, and M. Bayoumi, "A reversible-logic based architecture for long short-term memory (LSTM) network," in 2021 IEEE International Symposium on Circuits and Systems (ISCAS), 2021, p. 2.

[9] S. Selvin, R. Vinayakumar, E. A. Gopalakrishnan, V. K. Menon, and K. P. Soman, "Stock price prediction using LSTM, RNN and CNN-sliding window model," in 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), 2017, p. 1.

[10] S. Siami-Namini and A. S. Namin, "Forecasting economics and financial time series: ARIMA vs. LSTM," arXiv [cs.LG], 2018.

[11] P. Srivastava, "Long short term memory," Analyticsvidhya.com, 10-Dec-2017. [Online]. Available: https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm. [Accessed: 08-Sep-2021].

[12] G. Philipp, D. Song, and J. G. Carbonell, "The exploding gradient problem demystified - definition, prevalence, impact, origin, tradeoffs, and solutions," arXiv [cs.LG], p. 2, 2017.

[13] Y. Zhao, R. Yang, G. Chevalier, R. C. Shah, and R. Romijnders, "Applying deep bidirectional LSTM and mixture density network for basketball trajectory prediction," Optik (Stuttg.), vol. 158, pp. 266–272, 2018.

[14] "Understanding LSTM Networks," Github.io. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Accessed: 10-Sep-2021].

[15] J. Qiu, B. Wang, and C. Zhou, "Forecasting stock prices with long-short term memory neural network based on attention mechanism," PLoS One, vol. 15, no. 1, p. e0227222, 2020.

[16] T. Fischer and C. Krauss, "Deep learning with long short-term memory networks for financial market predictions," Eur. J. Oper. Res., vol. 270, no. 2, pp. 654–669, 2018.

[17] Q. Yu, K. Wang, J. O. Strandhagen, and Y. Wang, "Application of long short-term memory neural network to sales forecasting in retail—A case study," in Lecture Notes in Electrical Engineering, Singapore: Springer Singapore, 2018, pp. 11–17.

[18] D. Soutner and L. Müller, "Application of LSTM neural networks in language modelling," in Text, Speech, and Dialogue, Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 105–112.

[19] M. Mensio, E. Bastianelli, I. Tiddi, and G. Rizzo, "A multi-layer LSTM-based approach for robot command interaction modeling," arXiv [cs.CL], 2018.

[20] N. Zhang, S.-L. Shen, A. Zhou, and Y.-F. Jin, "Application of LSTM approach for modelling stress–strain behaviour of soil," Appl. Soft Comput., vol. 100, no. 106959, p. 106959, 2021.

[21] C. Kühnert, N. M. Gonuguntla, H. Krieg, D. Nowak, and J. A. Thomas, "Application of LSTM networks for water demand prediction in optimal pump control," Water (Basel), vol. 13, no. 5, p. 644, 2021.

[22] S. Santhanam, "Context based Text-generation using LSTM networks," arXiv [cs.CL], 2020.

[23] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional LSTM and other neural network architectures," Neural Netw., vol. 18, no. 5–6, pp. 602–610, 2005.

[24] Y. Ning et al., "Learning cross-lingual knowledge with multilingual BLSTM for emphasis detection with limited training data," in 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2017.

[25] S. Pardeshi, "CNN-LSTM architecture and image captioning - analytics Vidhya - medium," Analytics Vidhya, 23-Nov-2019. [Online]. Available: https://medium.com/analytics-vidhya/cnn-lstm-architecture-and-image-captioning-2351fc18e8d7. [Accessed: 07-Sep-2021].

[26] S. Skúli, "How to generate music using a LSTM neural network in keras," Towards Data Science, 07-Dec-2017. [Online]. Available: https://towardsdatascience.com/how-to-generate-music-using-a-lstm-neural-network-in-keras-68786834d4c5. [Accessed: 07-Sep-2021].

[27] Shkarupa, Yaroslav and Mencis, Roberts and Sabatelli, Matthia, Ed., Offline Handwriting Recognition Using LSTM Recurrent Neural Networks, vol. 1. THE 28TH BENELUX CONFERENCE ON ARTIFICIAL INTELLIGENCE November 10-11, 2016, Amsterdam (NL).

[28] J. Brownlee, "LSTMs for human activity recognition time series classification," Machinelearningmastery.com, 23-Sep-2018. [Online]. Available: https://machinelearningmastery.com/how-to-develop-rnn-models-for-human-activity-recognition-time-series-classification/. [Accessed: 10-Sep-2021].

[29] P. M. Sosa, "Twitter Sentiment Analysis using combined LSTM-CNN Models," 2017.

[30] M. Rana, M. M. Uddin, and M. M. Hoque, "Effects of activation functions and optimizers on stock price prediction using LSTM recurrent networks," in Proceedings of the 2019 3rd International Conference on Computer Science and Artificial Intelligence, 2019.

[31] B. Ding, H. Qian, and J. Zhou, "Activation functions and their characteristics in deep neural networks," in 2018 Chinese Control And Decision Conference (CCDC), 2018, pp. 1836–1841.

[32] C. Gulcehre, M. Moczulski, M. Denil, and Y. Bengio, "Noisy Activation Functions," arXiv [cs.LG], pp. 3059–3068, 2016.

[33] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, 2010, vol. 9, pp. 249–256.

[34] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, “Efficient BackProp,” in Lecture Notes in Computer Science, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 9–48.

[35] P. Lennie, “The cost of cortical computation,” Curr. Biol., vol. 13, no. 6, pp. 493–497, 2003.

[36] V. Nair and G. E. Hinton, “Rectified Linear Units Improve Restricted Boltzmann Machines,” 2010, pp. 807–814.

[37] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks",” 2012, pp. 1097–1105,.

[38] M. D. Zeiler, M. Ranzato, and R. Monga, “On rectified linear units for speech processing",” 2013, pp. 3517–3521,.

[39] R. K. Srivastava, J. Masci, S. Kazerounian, F. Gomez, and J. Schmidhuber, “Compete to compute",” 2013, pp. 2310–2318,.

[40] K. C. Naik, Stock-MArket-Forecasting. [Online].Available: https://github.com/krishnaik06/Stock-MArket-Forecasting.[Accessed: 26-Aug-2021]

[41] “sklearn.preprocessing.MinMaxScaler — scikit-learn 0.24.2 documentation,” Scikit-learn.org. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html. [Accessed: 10-Sep-2021].

[42] E. Culurciello, “The fall of RNN / LSTM - towards data science,” Towards Data Science, 13-Apr-2018. [Online]. Available: https://towardsdatascience.com/the-fall-of-rnn-lstm-2d1594c74ce0. [Accessed: 10-Sep-2021].

[43] D. Vilares and C. Gómez-Rodríguez, “A non-projective greedy dependency parser with bidirectional LSTMs,” arXiv [cs.CL], 2017.

[44] V. M. Posts, “LSTM is dead, long live Transformers,” Sea-adl.org, 03-Dec-2019. [Online]. Available: https://sea-adl.org/2019/12/03/lstm-is-dead-long-live-transformers/. [Accessed: 10-Sep-2021].

[45] A. Zeyer, P. Bahar, K. Irie, R. Schluter, and H. Ney, “A comparison of transformer and LSTM encoder decoder models for ASR,” in 2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU), 2019.

[46] “A beginner’s guide to LSTMs and recurrent neural networks,” Pathmind.com. [Online]. Available: https://wiki.pathmind.com/lstm. [Accessed: 01-Sep-2021].

[47] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio, “How to construct deep recurrent neural networks,” arXiv [cs.NE], 2013.