Recurrent Neural Networks (RNN) for Time Series Prediction namely for Stock Market Data

Harichandana Neralla
HXN210036
The University of Texas Dallas

Mrinalini Raghavendran
MXR210111
The University of Texas Dallas

Vamshi Krishna Agiri
VXA200029
The University of Texas Dallas

Sai Teja Reddy Komatireddy
SXK210130
The University of Texas Dallas

***Abstract*—This paper presents a time series stock prediction model using Long Short-Term Memory (LSTM) on the SP500 dataset. The primary objective is to predict future stock prices and trends by leveraging historical data. The proposed LSTM model demonstrates promising results in predicting stock prices with improved accuracy compared to traditional linear models. A comprehensive study of the LSTM architecture, its advantages, and a comparison with other techniques are presented. The results obtained are analyzed in detail, and potential improvements and future work are discussed.**

I. Introduction and Background Work

Financial analysis must include making accurate stock market forecasts because they can aid investors in choosing where to put their money. To predict stock values, a variety of algorithms and methodologies are utilized, from conventional linear regression models to cutting-edge machine learning techniques including Support Vector Machines, Neural Networks, and Deep Learning architectures.

For time series prediction problems, recurrent neural networks (RNNs), a form of machine learning method, have gained popularity. They are able to capture temporal dependencies, which are essential for predicting stock values, which explains why. It has been demonstrated that the RNN architecture known as Long Short-Term Memory (LSTM) is particularly good at simulating long-term dependencies.

Compared to standard RNNs, LSTM models are better at capturing the relationship between past and current inputs, which is important for forecasting stock prices. LSTM models have been effectively used in a variety of fields, including time series forecasting, speech recognition, and natural language processing. In the context of financial analysis, LSTM models are particularly useful for predicting stock prices and making informed investment decisions.

II. Data and Data Preprocessing

The S&P 500 dataset used in this study consists of daily stock prices from 2010 to 2021. The dataset was preprocessed to prepare it for model training and evaluation. The S&P 500 dataset is a collection of daily historical stock price data for the companies listed on the S&P 500 index, which is a stock market index that measures the stock performance of 500 large companies listed on the stock exchanges in the United States. The dataset typically includes information such as the date, opening price, closing price, highest and lowest prices for the day, volume of shares traded, and other financial metrics.
The dataset was preprocessed to prepare it for model training and evaluation. The preprocessing steps involved selecting the 'Close' column from the dataset, which represents the daily closing price. The data was then scaled using the MinMaxScaler to normalize the values between 0 and 1. This scaling is necessary as stock prices can vary significantly, and normalization ensures that the model can learn effectively from the data.

III. Theoretical and Conceptual Study of LSTM

The LSTM is a sort of recurrent neural network designed to address the vanishing gradient problem that happens when deep neural networks are trained. Memory cells, input gates, forget gates, and output gates comprise the LSTM architecture. Memory cells store and update the network's internal state, whereas gates manage the flow of information.

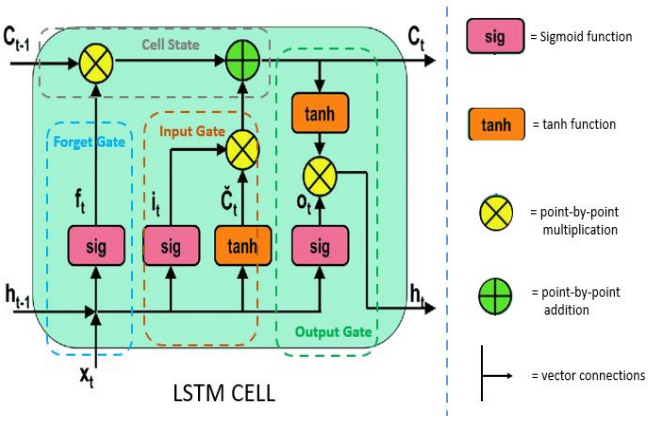

Fig. 1. LSTM Network Model Architecture.

A. Cell State

The cell state is a key concept in the Long Short-Term Memory (LSTM) architecture. It acts as an information highway that runs through the entire LSTM network and carries

information across time steps. The cell state is designed to allow important information to flow through the network while preventing less important information from interfering with the model's predictions.

At each time step in the LSTM network, the current input is combined with the previous cell state and the previous hidden state to compute a new "candidate" cell state value. This candidate value is then passed through a gating mechanism that decides which information to retain and which to discard. The gating mechanism is composed of three gates: the input gate, the forget gate, and the output gate.

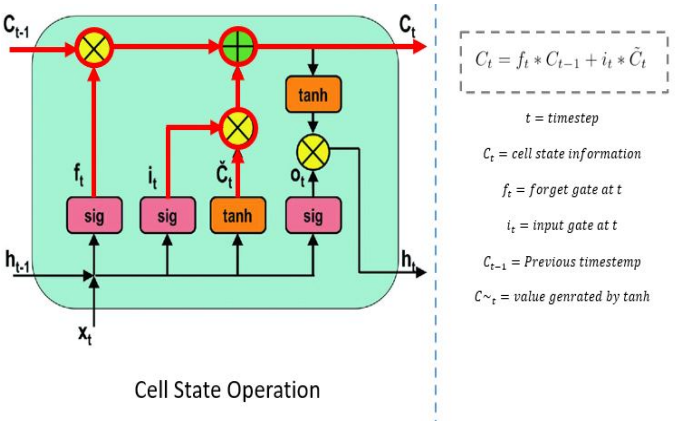

Fig. 2. Cell State.

B. Input Gate

The input gate determines the extent to which new information from the current input should be stored in the memory cell. It utilizes a sigmoid activation function that produces values between 0 and 1. A value close to 0 implies that the new information should be ignored, while a value close to 1 signifies that the new information should be stored in the memory cell. The input gate can be represented as:

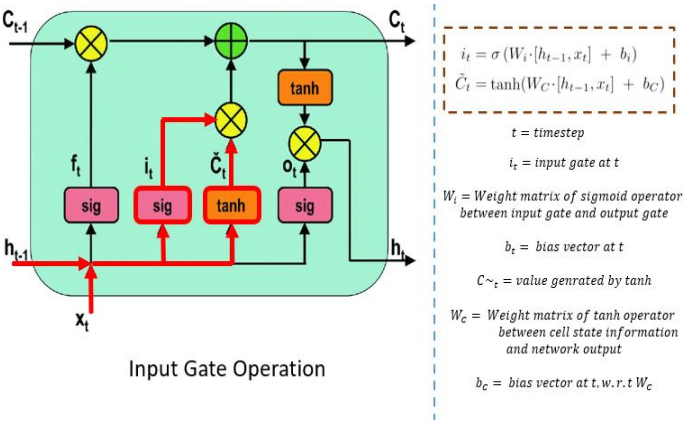

Fig. 3. Input Gate.

C. Forget Gate

The forget gate determines which information from the previous memory cell state (C_(t-1)) should be retained or discarded. Similar to the input gate, the forget gate also uses a sigmoid activation function to produce values between 0 and 1. A value close to 0 indicates that the previous information should be forgotten, while a value close to 1 means that the information should be retained. The forget gate can be represented as:

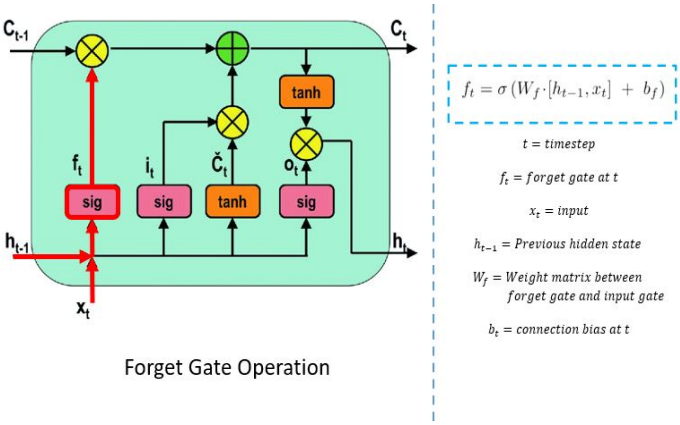

Fig. 4. Forget Gate.

D. Output Gate

The output gate controls the information flow from the memory cell to the hidden state (h_t), which is used for predictions and as input for the next LSTM cell in the sequence. The output gate also uses a sigmoid activation function to produce values between 0 and 1. These values determine the extent to which the information stored in the memory cell is used for predictions or passed on to the next LSTM cell. The output gate can be represented as:

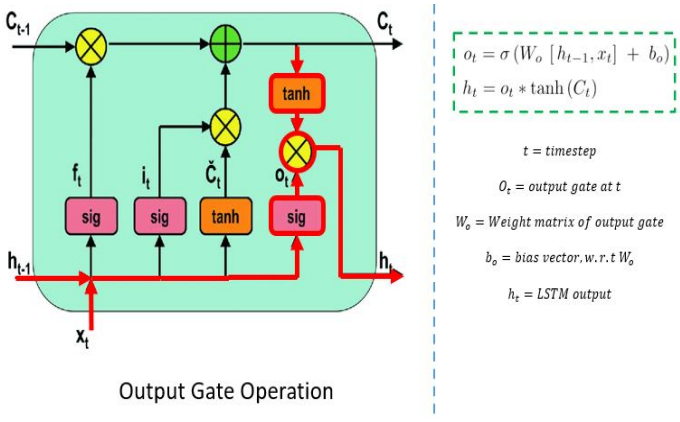

Fig. 5. Output Gate.

These gates work together to update the memory cell state (C_t) and the hidden state (h_t) at each time step, enabling

the LSTM model to capture long-term dependencies in time series data effectively. This capability is particularly useful for stock price prediction, as it allows the model to learn patterns and trends over time and make accurate predictions for future stock prices.

E. Forward and Backward Propagation

The forward and backpropagation functions in our project serve to update gradients and parameters using the chain rule, a fundamental concept in calculus. These two functions work in tandem to enhance the performance of our model by refining the weights and biases for each layer.

Forward propagation is the process of performing predictions based on the given dataset. It starts by computing the product of the inputs and their respective weights and then adds the bias component to this product. The initialized weights are applied to each of the gates (input, output, forget, and candidate gates), and the output is computed as the inputs are processed through these individual layers. This forward pass provides the predicted output for a given set of input data.

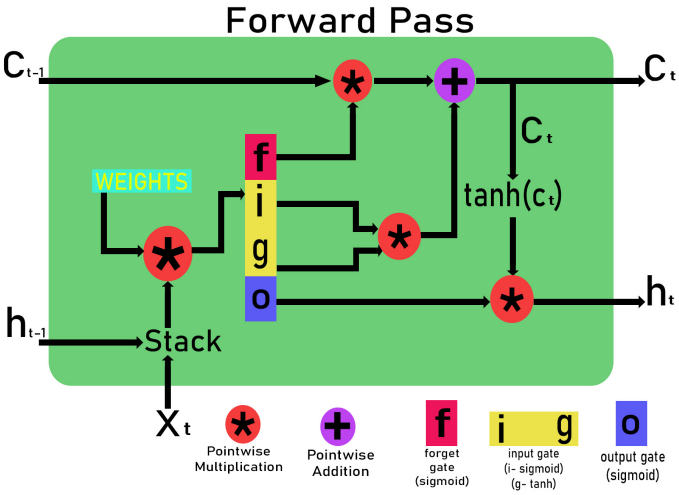

Fig. 6. Forward Propogation.

Backpropagation, on the other hand, is the process of calculating gradients with respect to a cost function, which in our case is the mean squared error (MSE) loss function. Using the gradient descent algorithm, backpropagation seeks to converge to a zero-gradient value, which represents the optimal parameters for our layers to multiply with the input dataset. Essentially, it determines how much each weight and bias should be adjusted to minimize the cost function. Both forward and backward propagation work together to calculate the gradients and update the weights for our parameters, ultimately improving the performance of the model over time as it learns from the training data.

To address the vanishing and exploding gradient problem, which can lead to poor model performance, we employ gradient clipping in our model. Gradient clipping constrains the range of gradient values, preventing them from becoming too small (vanishing) or too large (exploding). This technique helps maintain the stability of the model and ensures more reliable learning during the training process.

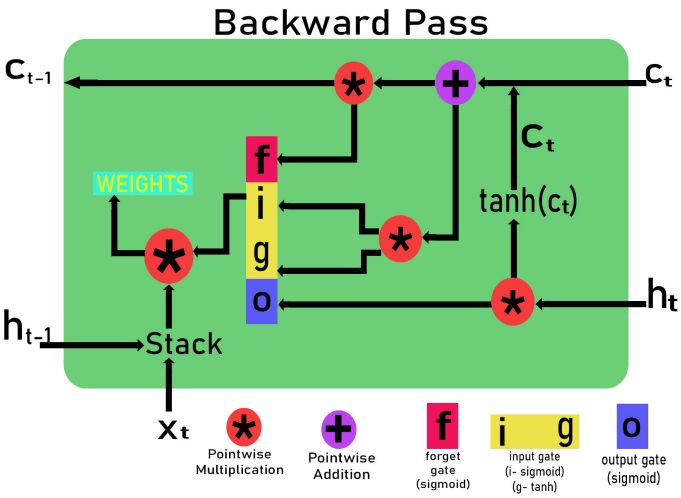

Fig. 7. Backward Propagation.

IV. Performance Metrics

To evaluate the performance of the LSTM model for stock price prediction, it is essential to use appropriate evaluation metrics that measure the model's accuracy and effectiveness. In this study, we consider the following evaluation metrics:

A. Mean Squared Error (MSE)

MSE measures the average squared difference between the predicted values and the actual values. It is another commonly used performance metric for regression problems. Lower MSE values indicate better prediction accuracy. MSE can be calculated as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

V. Hyper Parameters

A. Batch Size

Batch size refers to the number of training examples utilized in one forward/backward pass of a neural network during training. In the context of LSTM, batch size refers to the number of time series examples that are fed into the network at once. The batch size is an important hyperparameter that can affect the performance of the model during training.

A larger batch size can result in faster training times because the model is updated less frequently, but it can also result in larger memory requirements. On the other hand, a smaller batch size can result in slower training times because the model is updated more frequently, but it can also result in more stable convergence and better generalization.

B. Epoch

Epoch refers to one complete pass through the entire training dataset during the training process. In the context of LSTM, an epoch refers to one complete iteration through the entire time series dataset. During one epoch, the LSTM model processes all the training examples in the dataset and updates its weights based on the error between its predictions and the actual values.

A crucial hyperparameter that might impact the effectiveness of the LSTM model is the number of epochs. If there aren't enough epochs, the model might not have enough time to pick up on the underlying patterns in the time series data. However, if the number of epochs is too high, the model might overfit to the training set and underperform on fresh, untried data.

VI. Results and Analysis

The losses in the following tables represent the mean squared error cost function for Google stocks for 10 epochs. That is the number of times the dataset was fed in its entirety to the lstm model.

TABLE I
LOSSES OF EACH EPOCH FOR GOOGLE STOCK DATA

epoch no	Google
1	[[49.6650679]]
2	[[49.73504599]]
3	[[49.71534234]]
4	[[49.72536857]]
5	[[49.68720245]]
6	[[49.73142663]]
7	[[49.6998313]]
8	[[49.74947162]]
9	[[49.70067053]]
10	[[49.72053229]]

The losses in the following tables represent the mean squared error cost function for Apple stocks for 10 epochs.

TABLE II
LOSSES OF EACH EPOCH FOR APPLE STOCK DATA

epoch no	APPL
1	[[50.71246732]]
2	[[50.63975246]]
3	[[50.67475802]]
4	[[50.65317993]]
5	[[50.62083563]]
6	[[50.69085101]]
7	[[50.69661535]]
8	[[50.69405645]]
9	[[50.62382518]]
10	[[50.66092973]]

The losses in the following tables represent the mean squared error cost function for Amazon stocks for 10 epochs.

TABLE III
LOSSES OF EACH EPOCH FOR AMAZON STOCK DATA

epoch no	AMZN
1	[[50.53149688]]
2	[[50.53202]]
3	[[50.5421794]]
4	[[50.51971745]]
5	[[50.51709069]]
6	[[50.49376767]]
7	[[50.52107139]]
8	[[50.51721762]]
9	[[50.51795888]]
10	[[50.49997832]]

The losses in the following tables represent the mean squared error cost function for Fang stocks for 10 epochs

TABLE IV
LOSSES OF EACH EPOCH FOR FANG STOCK DATA

epoch no	FANG
1	[[49.52347855]]
2	[[49.50785337]]
3	[[49.5094276]]
4	[[49.60250704]]
5	[[49.54756956]]
6	[[49.48545683]]
7	[[49.49474871]]
8	[[49.47933666]]
9	[[49.49754299]]
10	[[49.53177073]]

The following plots represent mse loss against predicted value of y for four epochs for Apple stock data.

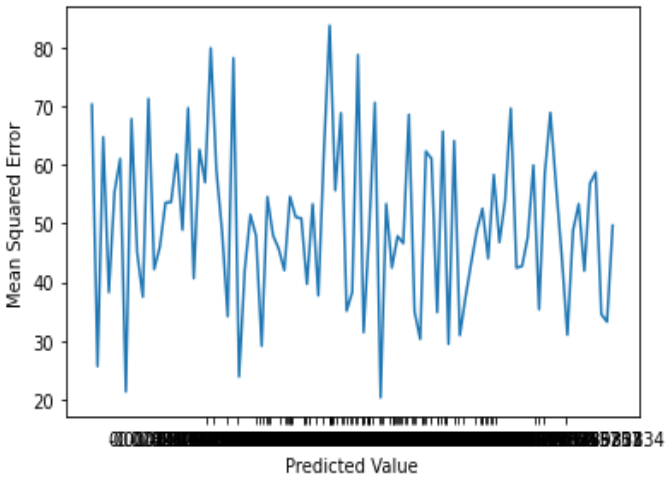

Fig. 8. Epoch 1.

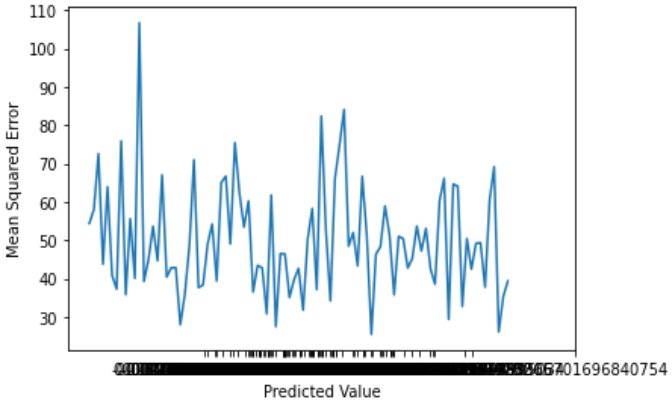

Fig. 9. Epoch 2.

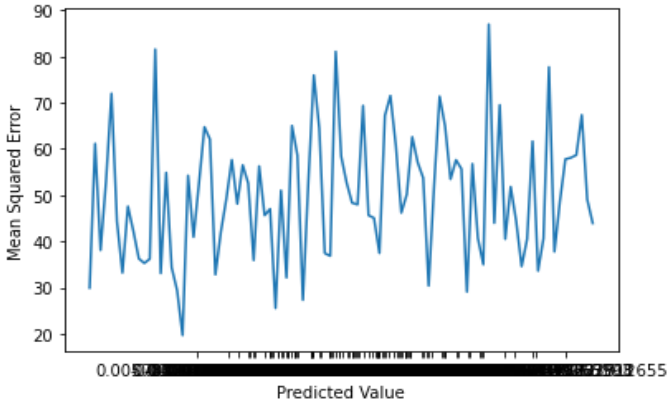

Fig. 10. Epoch 3.

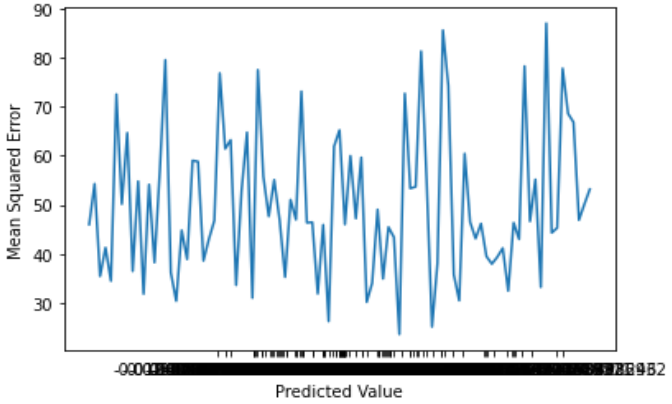

Fig. 11. Epoch 4.

REFERENCES

[1] https://www.turing.com/kb/comprehensive-guide-to-lstm-rnn
[2] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.
[3] pluralsight.com/guides/introduction-to-lstm-units-in-rnn
[4] https://vijay-choubey.medium.com/how-to-evaluate-the-performance-of-a-machine-learning-model-d12ce920c365
[5] https://www.geeksforgeeks.org/lstm-derivation-of-back-propagation-through-time