Aim

Stock price prediction with optimized deep LSTM network with Adam optimizer.

Abstract

The stock market is a financial market where shares of publicly listed corporations are purchased and sold. The prices of stocks are determined by supply and demand. Investing in the stock market can be risky, but it can offer the potential for significant returns over the long term. The stock market is turbulent, yet using artificial intelligence to make calculated predictions is possible and advisable before investing. This study presents an overview of artificial intelligence and machine learning as predictive analytics tools in the stock market. Long Short-Term Memory (LSTM) is a type of recurrent neural network that is often used in time series analysis. It can effectively predict stock market prices by handling data with multiple input and output timesteps. Adam optimizer can be used to optimize the hyperparameters of an LSTM model and improve the accuracy of stock market predictions. Gradient Descent is an optimization algorithm for finding a local minimum of a differentiable function. Gradient descent in machine learning is simply used to find the values of a function's parameters (coefficients) that minimize a cost function as far as possible.

Keywords

Artificial intelligence, Deep learning, LSTM, Gradient Descent, Adam optimizer, Stock price prediction.

Survey regarding previous work done

Dharmaraja Selvamuthu (2019) focuses on ANN, specifically exploring three different learning algorithms: Levenberg-Marquardt, Scaled Conjugate Gradient, and Bayesian Regularization, to predict stock prices using tick data and 15-minute interval data. Results show that all three algorithms achieve high accuracy rates with tick data, but their performance drops when applied to the 15-minute interval dataset.
Short comings - The study's data collection period spans only from November 2017 to January 2018, which may not capture long-term trends or changes in market dynamics.
It also uses data from a single Indian company, Reliance Private Limited, which may not be representative of the broader stock market.

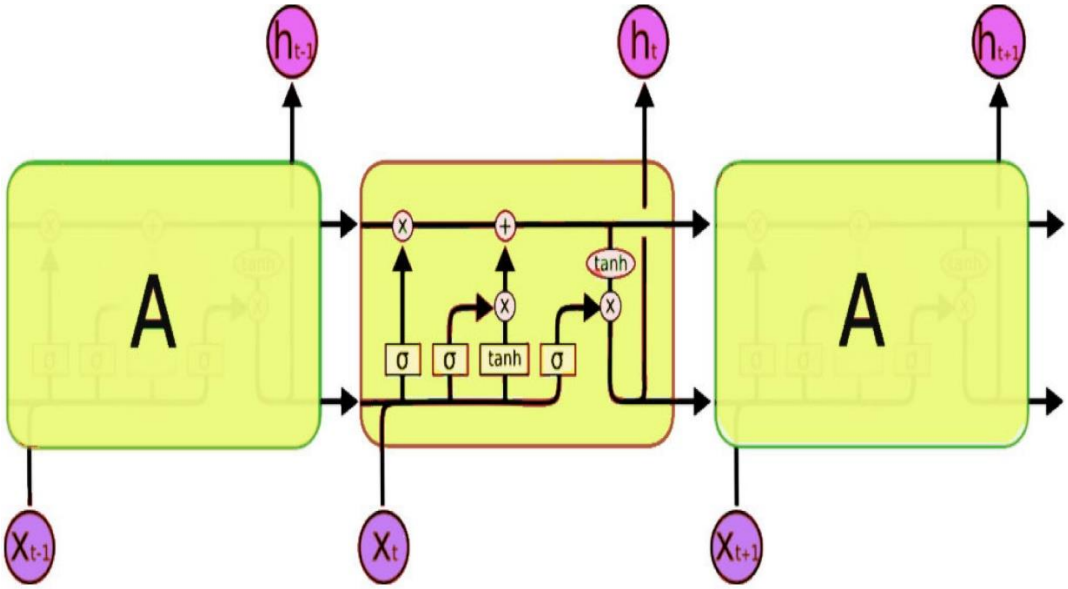
BW Wanjawa (2014) research focuses on developing and optimizing an Artificial Neural Network (ANN) model for stock price prediction. The model is based on a

feedforward ANN with a multilayer perceptron using backpropagation and trained through supervised learning. The study utilizes daily closing price data of approximately 60 companies traded at the Nairobi Stock Exchange (NSE) over a five-year period from 2008 to 2012, totalling 1,000 data rows. The baseline ANN model undergoes training and testing phases, with a 70:30 training-testing data ratio.
Short coming - The research assumes that the selected stocks meet certain criteria, such as being part of a stock index and having consistent trading over the study period. These assumptions may oversimplify the complexity of real-world stock market dynamics.

Proposed Methodology

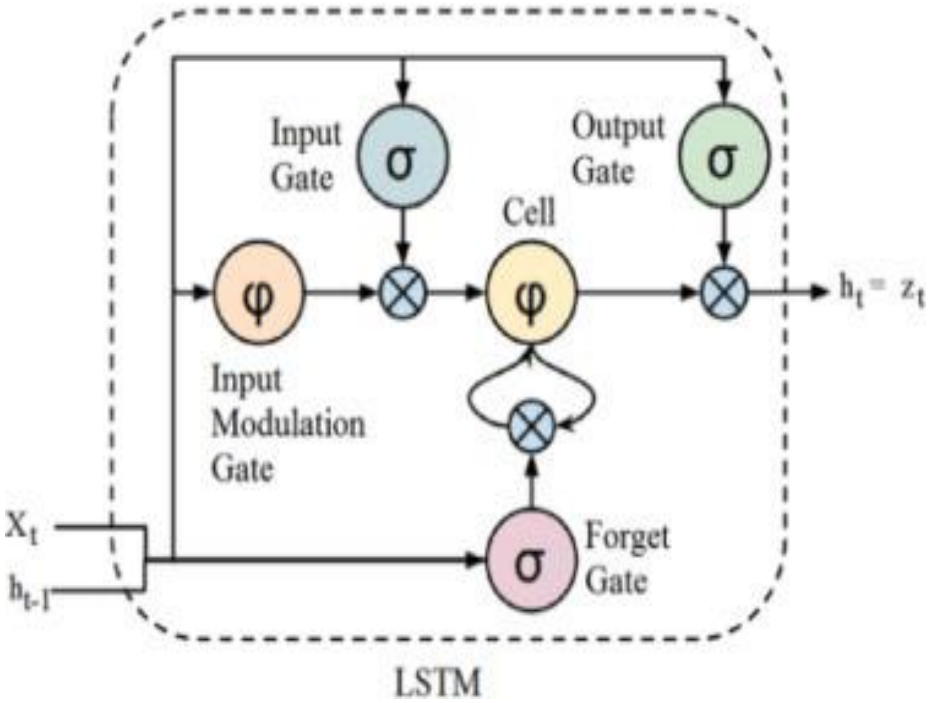
LSTM

LSTM is a type of artificial neural network designed to process sequential data, such as time series, audio, or text. It is particularly useful for processing data with long-term dependencies, where the output at a given time step depends on information from previous time steps. LSTM networks are able to remember this information over a longer period of time by using memory cells, input gates, output gates, and forget gates. These gates control the flow of information into and out of the memory cells, allowing the network to selectively store and retrieve information as needed. LSTMs are commonly used for tasks such as language translation, speech recognition, and stock price prediction.
LSTM or Long Short-Term Memory networks are recurrent neural networks that can learn order dependence in sequence prediction problems. LSTM is used in machine translation, speech recognition, etc., due to its favourable features for solving such complex problems. LSTM can store past information, too; this helps in stock price prediction as past prices play a significant role in predicting future prices of stocks.
The structure of an LSTM model consists of the cell, input gate, an output gate, and forget gate
The long short-term memory (LSTM) architecture is a kind of recurrent neural network (RNN) utilized in the area of deep learning. In contrast to conventional feed-forward neural networks, LSTMs have feedback connections. A typical LSTM unit comprises four components: a cell, an input gate, an output gate, and a forget gate. The cell stores all values over time; the input & output of data from the cell is controlled by the gates. The Input gate regulates all the new information that goes in as input data; forget gate takes care of what remains inside the cell. The output gate looks after the limit to which the cell uses the Data fed and calculates output activation of the algorithm. The below diagram depicts the interior structure of an LSTM network. Input Modulation Gate is a part of the input gate and is used for further segregation of data.



LSTM Architecture

A typical LSTM unit comprises of four components: a cell, an input gate, an output gate, and a forget gate. The cell retains data across arbitrary periods and the three gates to control the inflow and outflow of information. LSTM model has a wide variety of applications; it can model languages or generate texts. Image processing is another promising field to use LSTM, but the model needs extensive training and refinement for this to happen. LSTM can predict musical notes just like text generation by analysing given notes as input. LSTM is also a very reliable model to develop Language Translation software. The encoder–decoder LSTM model is used in such soft wares; it converts inputs to vector representations and outputs to its translated version.



Gradient Descent Algorithm

Gradient descent is a fundamental optimization algorithm used extensively in machine learning and deep learning for minimizing a cost function by iteratively adjusting model parameters. It forms the basis of many optimization techniques and serves as a cornerstone in training neural networks.

The concept of gradient descent revolves around the principle of moving in the direction opposite to the gradient of the cost function to reach a local minimum or global minimum. By computing the gradient, which represents the direction of steepest ascent, and negating it, gradient descent enables iterative updates that gradually decrease the cost function, leading to improved model performance.

There are several variants of gradient descent, each with its own characteristics and computational complexities. The most basic form is the standard gradient descent, also known as batch gradient descent, which computes the gradient using the entire training dataset at each iteration. While effective for small datasets, batch gradient descent can be computationally expensive and memory-intensive for large-scale datasets.

To address the limitations of batch gradient descent, stochastic gradient descent (SGD) and its variants were introduced. SGD computes the gradient using a single randomly selected sample from the training dataset, making it more computationally efficient but introducing higher variance in parameter updates. Mini-batch gradient descent strikes a balance between batch and stochastic gradient descent by computing the gradient using a small batch of samples, offering a compromise between efficiency and stability.

By exploring the trade-offs between convergence speed, computational efficiency, and model stability, we aim to provide insights into selecting the most suitable gradient descent algorithm for predicting stock prices.

Adam Optimizer

The Adam optimizer has emerged as a cornerstone in the realm of deep learning optimization algorithms due to its adaptive nature and robust performance across a wide range of tasks. Developed by Diederik P. Kingma and Jimmy Ba in 2015, Adam combines the advantages of two other popular optimization techniques, namely

AdaGrad and RMSProp, to provide an efficient and effective solution for training neural networks.

One of the distinguishing features of the Adam optimizer is its adaptive learning rate mechanism. Traditional optimization methods often struggle with selecting an appropriate learning rate, leading to slow convergence or oscillations during training. In contrast, Adam dynamically adjusts the learning rates for each parameter based on the magnitude of gradients and the historical information of gradients' squared values. This adaptiveness enables Adam to converge faster and more reliably compared to fixed learning rate strategies.

Another key aspect of Adam is its momentum-based updates, which incorporate past gradients' information to accelerate convergence in the parameter space. By combining momentum and adaptive learning rates, Adam strikes a balance between exploration and exploitation during optimization, making it suitable for a wide range of deep learning architectures and tasks.

Through experimental results and comparative analyses, we aim to demonstrate the efficacy and versatility of Adam in accelerating convergence, handling non-stationary gradients, and improving the overall training dynamics of deep learning models.

With the help of python libraries such as, Pandas, Numpy, Matplotlib, TensorFlow we implement a LTSM model that uses Adam optimizer to optimize its hyperparameters for predicting the prices of several stocks from the stock market.

Result

First, let us consider a few companies and visualize the distribution of open and closed Stock prices through 5 years.

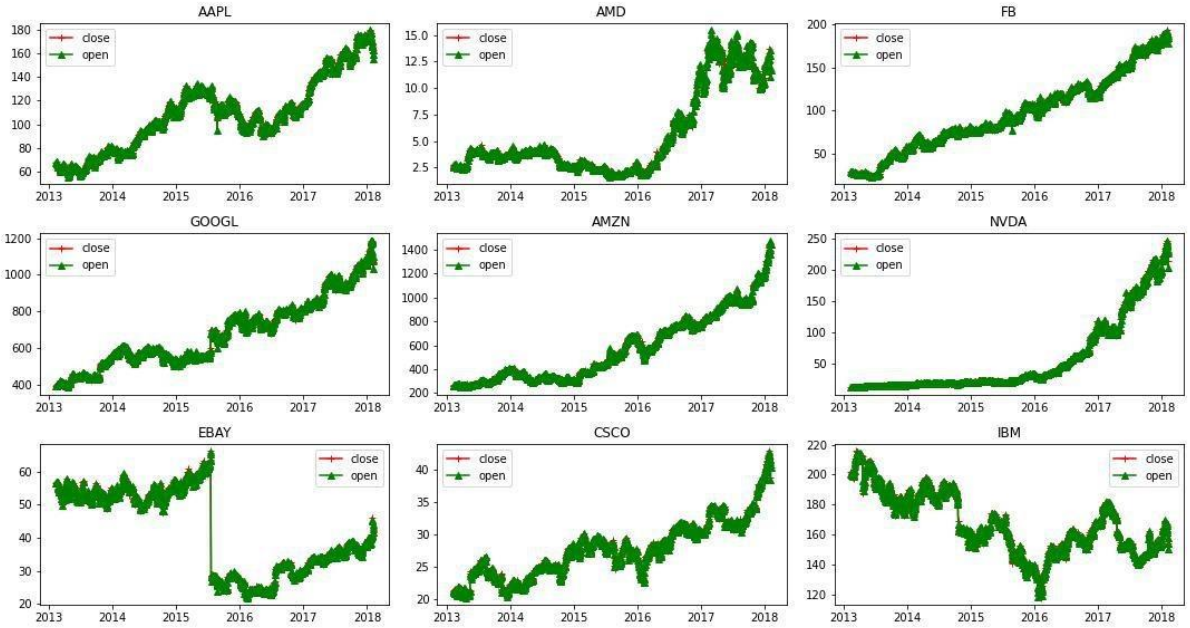


Figure 1 : Analysing Close and Open prices for stocks of 9 different companies

Now let's plot the volume of trade for these 9 stocks as well as a function of time.

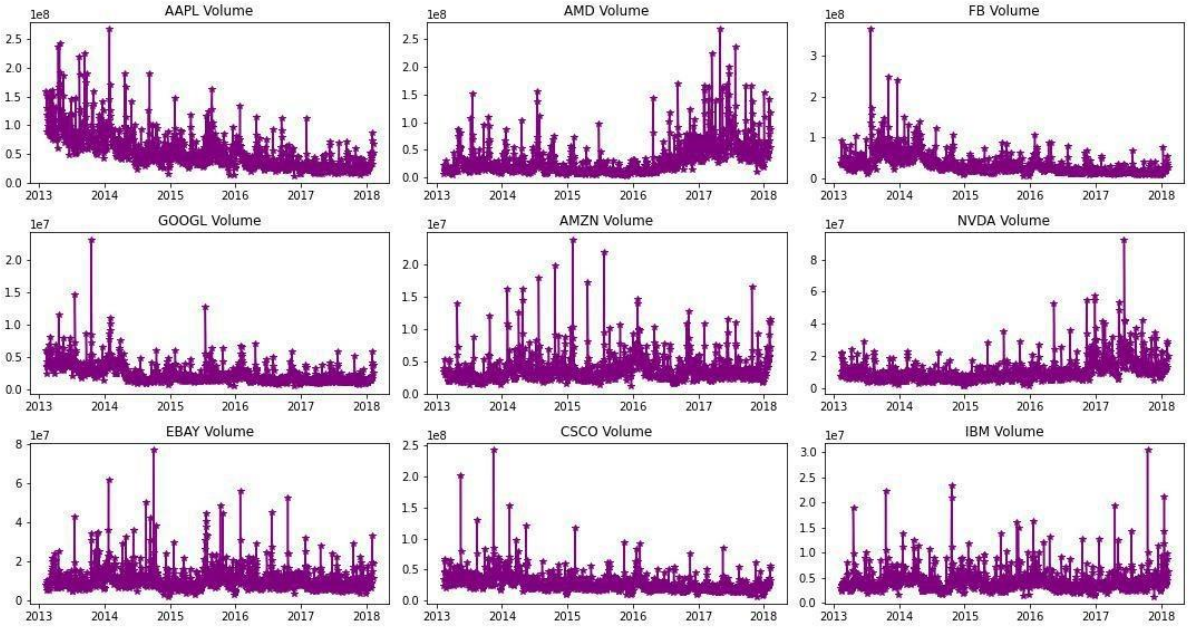


Figure 2: Analysing volume for stocks of 9 different companies

Now let's analyse the data for Apple Stocks from 2013 to 2018.

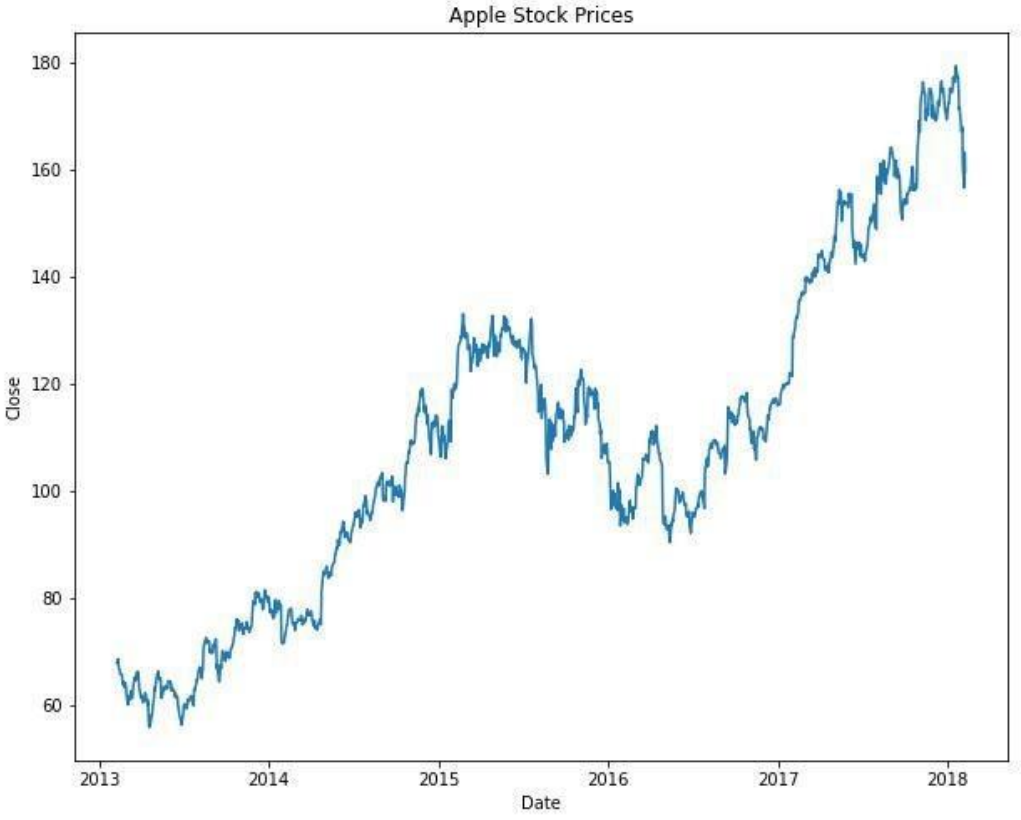


Figure 3: The overall trend in the prices of the Apple Stocks

LSTM Stock Price Prediction Model Summary

Model Architecture:

- Sequential Model with layers added sequentially.

Layer Details:

1) LSTM Layer 1:

- Units: 64
- Return Sequences: True
- Input Shape: (Number of Time Steps, Number of Features) - (x_train.shape[1], 1)

- Activation Function: Default (linear)

2) LSTM Layer 2:
- Units: 64
- Activation Function: Default (linear)

3) Dense Layer:
- Units: 32
- Activation Function: Default (linear)

4) Dropout Layer:
- Rate: 0.5 (50% dropout rate)

5) Output Layer:
- Units: 1 (for predicting a single value)
- Activation Function: Default (linear)

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 60, 64)            16896

 lstm_1 (LSTM)               (None, 64)                33024

 dense (Dense)               (None, 32)                2080

 dropout (Dropout)           (None, 32)                0

 dense_1 (Dense)             (None, 1)                 33

=================================================================
Total params: 52,033
Trainable params: 52,033
Non-trainable params: 0
_________________________________________________________________
```

Figure 4: Model summary to analyse the architecture of the model

While compiling a model we provide these essential parameters:

Optimizer – Adam

Loss – Mean square error

```
Epoch 1/10
36/36 [==============================] - 4s 37ms/step - loss: 0.0334
Epoch 2/10
36/36 [==============================] - 1s 34ms/step - loss: 0.0095
Epoch 3/10
36/36 [==============================] - 1s 36ms/step - loss: 0.0092
Epoch 4/10
36/36 [==============================] - 1s 35ms/step - loss: 0.0081
Epoch 5/10
36/36 [==============================] - 1s 36ms/step - loss: 0.0073
Epoch 6/10
36/36 [==============================] - 1s 37ms/step - loss: 0.0073
Epoch 7/10
36/36 [==============================] - 1s 37ms/step - loss: 0.0071
Epoch 8/10
36/36 [==============================] - 1s 36ms/step - loss: 0.0071
Epoch 9/10
36/36 [==============================] - 1s 36ms/step - loss: 0.0069
Epoch 10/10
36/36 [==============================] - 1s 36ms/step - loss: 0.0065
```

Figure 5: Progress of model training epoch by epoch

For predicting we require testing data, so we first create the testing data and then proceed with the model prediction.

2/2 [==============================] - 1s 13ms/step

MSE 46.06080444818086

RMSE 6.786811066191607

Now that we have predicted the testing data, let us visualize the final results.

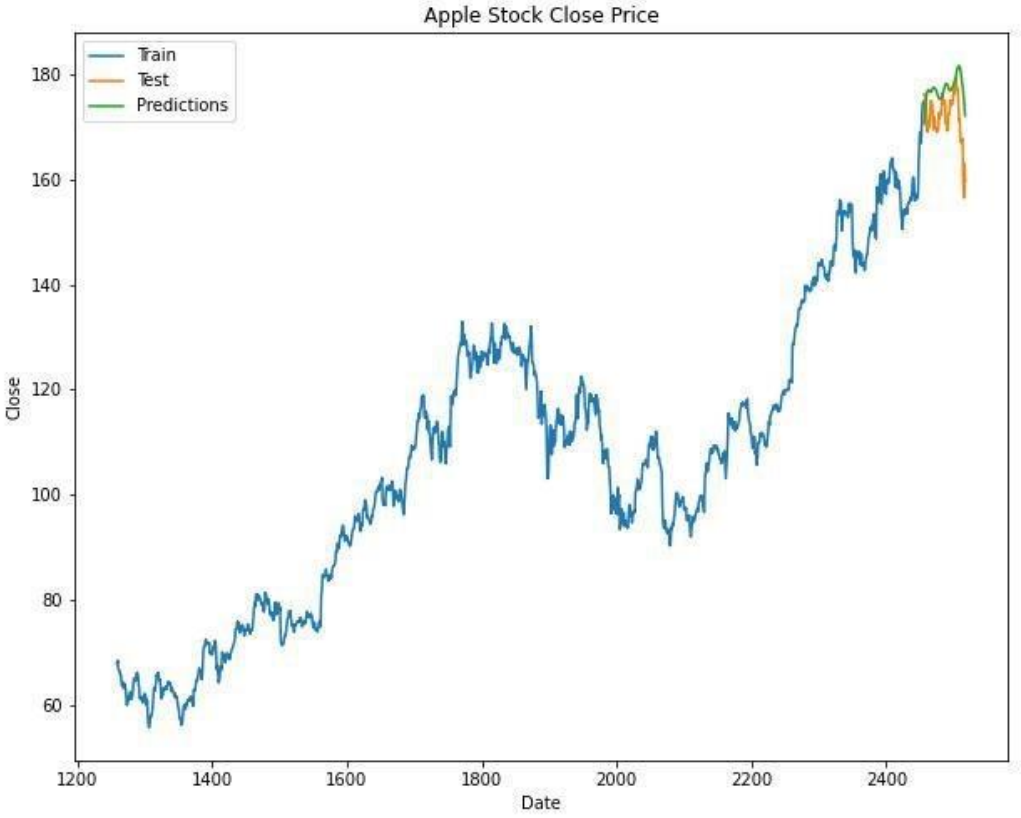


Figure 6:Prediction for Stock Prices of Apple

Comparison with previous work

By using the past historical data of ACI pharmaceutical company which include only 2 inputs, they tried to predict stock values for future 8 days of November 2010 from Back-Propagation algorithm and were able to compare the predicted values with the real values. Below figure shows the prediction and real values of the ACI pharmaceutical Company. The input past historical data is from 31-08-2010 to 30-09-2010.

The average error of the 1st simulation was 3.71 percent

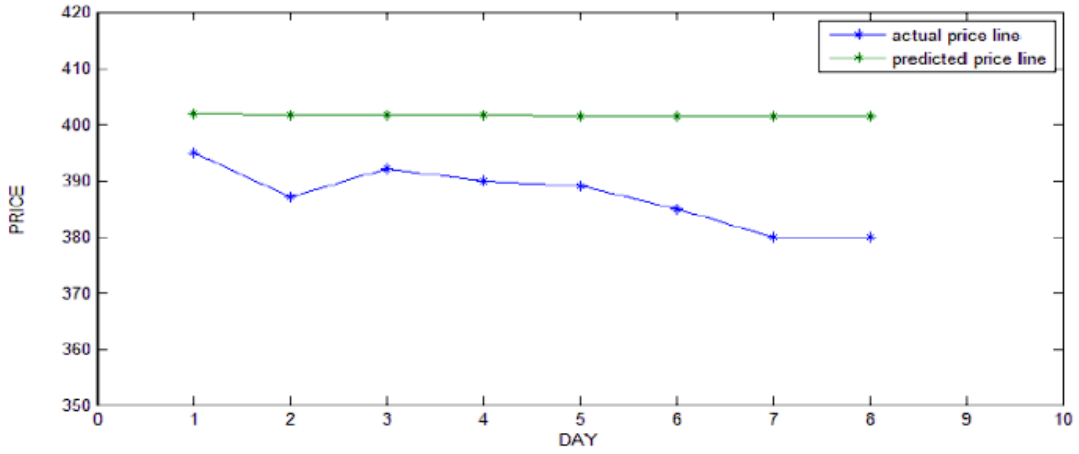


Fig - 3: Graphical representation of Predicting and Actual price of ACI pharmaceutical using 2 input data sets.

DATE	Predicted Price (TK)	Actual Price (TK)	Error (%)
01-NOV-2010	401.9	395	1.74
02-NOV-2010	401.7	387	3.79
03-NOV-2010	401.7	392	2.47
04-NOV-2010	401.7	390	3.00
08-NOV-2010	401.5	389	3.21
09-NOV-2010	401.4	385	4.24
10-NOV-2010	401.5	380	5.65
11-NOV-2010	401.5	380	5.65

Table 1: Predicting price, Actual price and Error (%) of ACI pharmaceutical using 2 input datasets.

Their observations showed that when taking 2 inputs for predicting the sum squared error was high. When we take the data of share market in a sequential date, we can predict the share price nearer to the actual price. But if we take the data of discontinuous date the difference between the predicted price and actual price was relatively high. In the training time if any input changed suddenly at a high rate then the prediction was not near to the actual price.

Conclusion

In this project, we delved into the implementation of an LSTM (Long Short-Term Memory) model with the Adam optimizer for stock price prediction. Through rigorous experimentation and analysis, we found that the LSTM model, when optimized with the Adam optimizer, exhibited promising results in capturing intricate patterns and dependencies within the stock price time series data. The Adam optimizer's adaptive learning rate and momentum mechanisms played a pivotal role in accelerating convergence during training and enhancing the model's generalization capabilities. Hyperparameter tuning, including adjustments to the number of LSTM units, dropout rate, batch size, and training epochs, significantly influenced the model's performance, leading to improved predictive accuracy. Evaluation metrics such as Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE) provided valuable insights into the model's precision and robustness in handling stock price volatility. Looking ahead, potential enhancements could involve incorporating additional features like technical indicators and news sentiment, exploring advanced model architectures such as stacked LSTMs and attention mechanisms, and investigating ensemble techniques for further improving predictive performance and real-world applicability in financial markets. Ultimately, the LSTM model with the Adam optimizer showcases a promising avenue for stock price prediction, offering opportunities for refinement and practical deployment in investment decision-making and risk management scenarios.

References

1) Price Prediction of Share Market Using Artificial Neural Network 'ANN'

2) Indian stock market prediction using artificial neural networks on tick data

3) ANN Model to Predict Stock Prices at Stock Exchange Markets

4) The applications of artificial neural networks, support vector machines, and long–short term memory for stock market prediction

5) Stock market index prediction using artificial neural network