COEN 240 Final Report

Amazon Stock Price Prediction using Recurrent Neural Networks

Spring 2022

Lalitha Madhuri Tanikella - 00001636451
Bhanu Prakash Naredla - 00001630571

TABLE OF CONTENTS

TABLE OF FIGURES

1. ABSTRACT

In this modern world, as data is growing day by day at a rapid pace, the complexity to predict it grows too. One of the massive sources for such big data is the “Stock Market” where day in day out trading of company shares takes place. Stock market is volatile in nature; it keeps fluctuating based on the company’s performance, market value, past trends, etc. However, if there was a way for traders to predict the stock price, their lives would be much simpler and easier. And which other branch of study would tackle this problem better than Machine Learning. Machine Learning can be applied to a wide range of crucial applications. In this particular project, we’re going to use “Recurrent Neural Networks” particularly focusing on “Long Short Term Memory(LSTM)” to solve this problem.Machine Learning algorithms to predict the future stock prices of “Amazon” company. We are going to view stock data as time series i.e, we are going to consider the past 5 year’s stock prices(including other features) of Amazon and try to predict its future value using Long Short Term Memory(LSTM) - a type of Recurrent Neural Network. Neural Networks are extremely powerful and versatile and can therefore be used to solve complex problems in Machine Learning like Speech Recognition, Handwriting Recognition, Stock Exchange Recognition, etc. Specifically, Recurrent Neural Networks are famous for handling time series data such as weather forecasting, house prices, stock prices, etc. We use LSTM where each memory cell learns what previous information to forget, what new relevant information to keep, and what information should be passed on to the next cell thereby solving the ‘Vanishing Gradient’ problem that RNNs face.

2. MOTIVATION & PREPARATION

2.1 MOTIVATION

Machine Learning is defined as a field of study that involves designing algorithms that learn from complex data and try to predict future outcomes. The goal of our project is to predict the future stock prices of Amazon using previous year's data thereby trying to analyze how the future stock market for Amazon might look like. We're going to perform this analysis using LSTM - one of the well-known techniques to deal with time-series data.

2.2 PREPARATION

2.2.1 Platform Used:

As part of this project, we used Google Colab to run our piece of code and perform analysis. This platform allows users to easily execute any kind of Python code without any hassle of downloading external libraries. It is well suited for applications that involve Machine Learning, Data Analysis, etc.

2.2.2 Tools Used:

We used the following tools as part of this project:

- *Pandas* - To load the dataset
- *Matplotlib* - To plot our results and check the accuracy
- *Tensorflow.Keras* - To train our model using SimpleRNN and LSTM
- *Sklearn* - To perform Feature Scaling, split train and test data, and calculate mean squared error and mean absolute percentage error.

3. ALGORITHM AND ARCHITECTURE

3.1 NEURAL NETWORK

Neural Networks also called as Artificial Neural networks (ANNs) are a set of algorithms designed to recognize patterns. Working of neural networks is similar to the working of a human brain, mimicking the way that biological neurons signal to one another. Artificial neural networks (ANNs) are made of node layers, containing an input layer, one or more hidden layers, and an output layer.

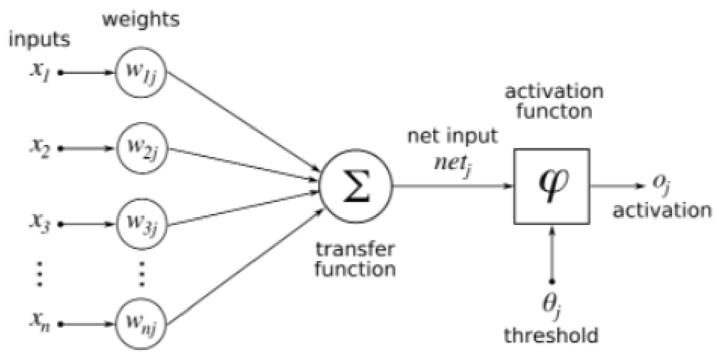


Fig 3.1 Artificial neuron structure

The nodes in the ANNs are also called artificial neurons. Artificial neurons connect to another and have an associated weight and threshold. If the output from that neuron is above the specified threshold, the neuron is activated and the output is passed from that neuron to other connected neurons, Fig 3.1 shows the structure of an artificial neuron. Output from a node layer is passed to its next layer or input to the current layer is received from the preceding layer, Fig 3.2 shows the architecture of an Artificial Neural Network (ANN) where each node is an artificial neuron.

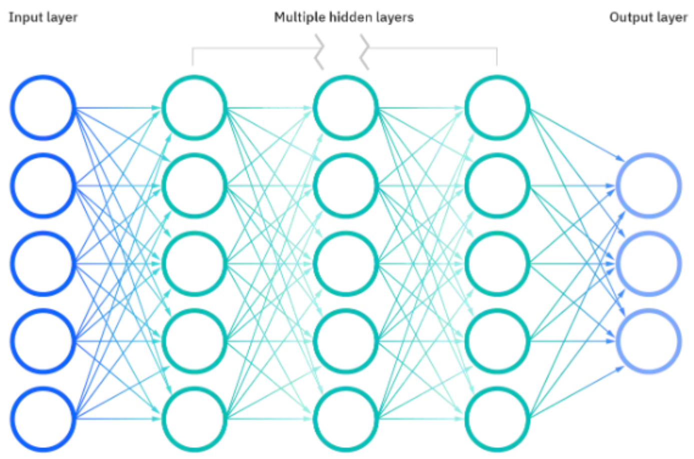


Fig 3.2 Artificial neural network

3.2 FORWARD AND BACKWARD PROPAGATION

3.2.1 Forward Propagation

Data flows only in the forward direction from the input layer through hidden layers and to the output layer. By this we get the output of the model, then we check the accuracy of the model and calculate the error.

3.2.2 Backward Propagation

Back propagation method is used to train neural networks. Once we find the accuracy and get the error after forward propagation, we back propagate the error to the network for updating the weights.

We find the partial derivatives of the error with respect to the weights by going backward through the network. This partial derivative is now multiplied by the learning rate to calculate the new weights. Thus, the neural network is learning during the training process.

3.3 RECURRENT NEURAL NETWORK

Recurrent neural network (RNN) is similar to a traditional neural network but has an internal memory. RNN is recurrent in nature as it performs the same computation for every input data, the output of the current input depends on the last computation. For better understanding, in Fig

3.3 First it takes X(0) from the input sequence and outputs h(0). Now, h(0) and X(1) will be the input to the next step and so on.

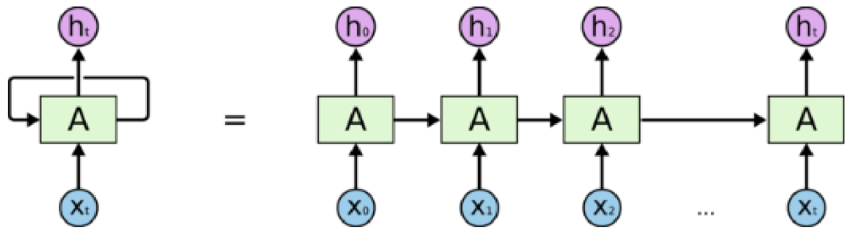


Fig 3.3 An unrolled recurrent neural network

RNN has internal memory, it memorizes previous data. While making a decision, it considers the current input and also what it has learned from the previous inputs. Output from the previous step is fed as input to the current step. Equations for Fig 3.4 are as follows

At Timestep(t), $S_t = ActivationFunction(U * X_t + W * S_{t-1})$

$y_t = Softmax(V * S_t)$

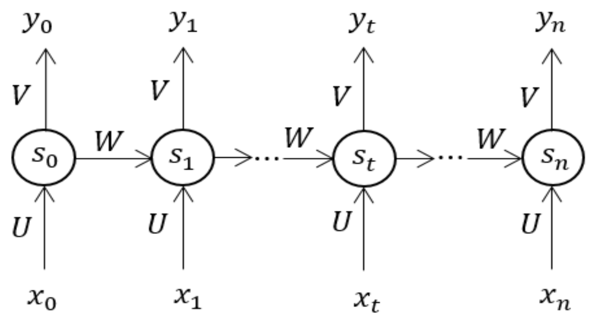


Fig 3.4 recurrent neural network

Now, we use backpropagation through time to calculate new weights and update.

3.3.1 Backpropagation Through Time (BPTT)

Applying a backpropagation algorithm to a RNN with time series data as input is backpropagation through time. As discussed in section 3.2.2 we calculate the partial derivatives of the error with respect to inputs, take a look at the Fig 3.5 for reference.

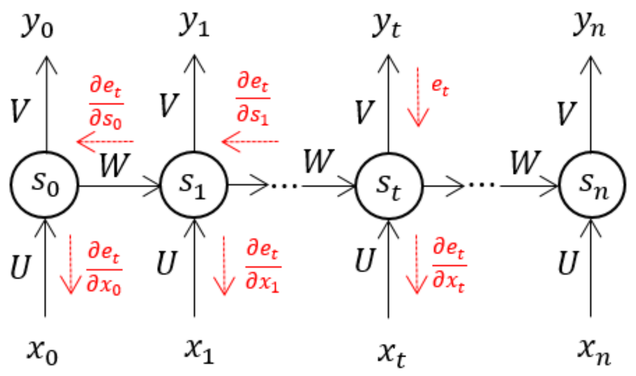

y_0
y_1
y_t
y_n
V
$\frac{\partial e_t}{\partial s_0}$
$\frac{\partial e_t}{\partial s_1}$
e_t
W
s_0
s_1
s_t
s_n
U
$\frac{\partial e_t}{\partial x_0}$
$\frac{\partial e_t}{\partial x_1}$
$\frac{\partial e_t}{\partial x_t}$
x_0
x_1
x_t
x_n

Fig 3.5 backpropagation through time in RNN

3.3.2 Vanishing/Exploding gradient problem

Since we are multiplying previous state by weight matrix over and over again, the gradient may be scaled up or down depending on the largest singular value of the weight matrix. If the singular value is greater than one then we'll face an exploding gradient problem, giving higher priority to the past data. If the singular value is less than one then we'll face an vanishing gradient problem, not considering past data from long back. Thus, in either case the weight matrix isn't optimized as expected.

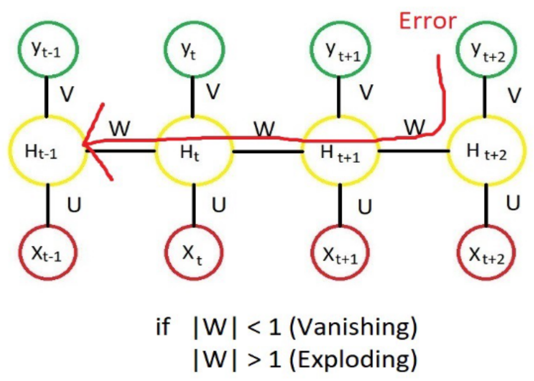


Fig 3.6 Vanishing/Exploding gradient problem in RNN

Long-Short Term Memory Networks (LSTM) won't have these problems.

3.4 LONG-SHORT TERM MEMORY NETWORKS (LSTM)

Long-Short Term Memory network (LSTM) is a kind of recurrent neural network which is explicitly designed to handle long-term dependency problem. Remembering information for a long period of time is practically their natural behavior. As LSTM is a kind of RNN, it also has a chain of repeating modules of neural network but the repeating module just has a different structure.

3.4.1 Structure of LSTM Cell

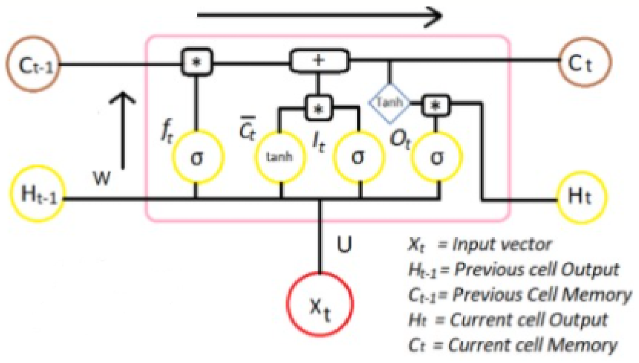


Fig 3.7 Structure of LSTM Cell

Here previous cell memory, previous cell output and current input is passed as input to a LSTM cell. There are four gates namely forget gate, input gate, update gate and output gate interacting in a special way to yield current cell memory and current cell output which is passed as input to the next cell down the chain. I'll explain the working of these four gates in the following sections.

3.4.2 Core Idea Behind LSTMs

The key to LSTMs is the cell state, the horizontal line running through the top in Fig 3.7, it runs straight down the entire chain with only some minor interactions.

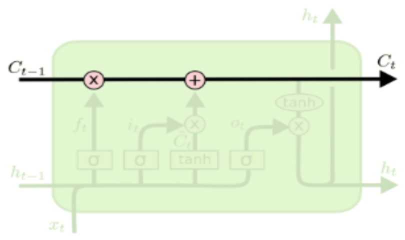


Fig 3.8 LSTM cell state

LSTM has the ability to add or remove information to the cell state using gates. Gates are used to optionally let through the information from the input to the cell state. It comprises a sigmoid neural net layer and a pointwise multiplication operation as shown in Fig 3.9 .

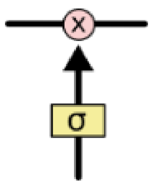


Fig 3.9 gate in an LSTM cell

The sigmoid layer outputs the number between zero and one, suggesting how much of information from each component be let through. A value of zero means nothing to let through, while a value of one means everything to let through. An LSTM cell has three of these gates, to protect and control the cell state.

3.4.3 Working of LSTM

3.4.3.1 Forget Gate

Forget gate layer looks at the h_{t-1} and x_t and outputs the number between zero and one for each number in the cell state C_{t-1}, representing what information we are going to throw away from the cell state.

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

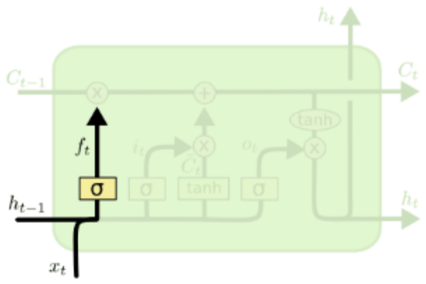


Fig 3.10 Forget gate in LSTM cell

3.4.3.2 Input Gate

Input gate has two parts. First, a sigmoid layer helps in deciding which new values must be updated in the cell state. Next, a tanh layer creates a vector with new values which can be added to the cell state. Thus, we'll combine these two to update cell state.

$$i_t = \boldsymbol{\sigma}(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\acute{C}_t = tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$

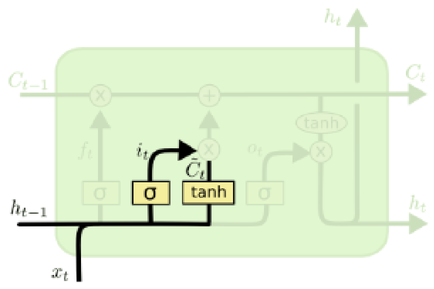


Fig 3.11 Input gate in LSTM cell

3.4.3.3 Update Gate

With the help of outputs from the forget gate and input gate, we can now update the previous cell state C_{t-1} to make it the current cell state C_t. To forget the things we decided to forget earlier, we multiply the previous cell state by output of the forget gate f_t. Then we add the new information by multiplying $i_t * Ć_t$. So, the update equation comes as $C_t = f_t * C_{t-1} + i_t * Ć_t$.

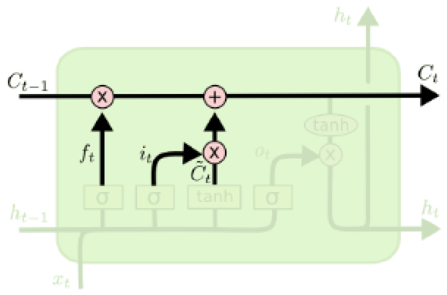


Fig 3.12 Updating previous cell state

3.4.3.4 Output Gate

Now we decide what we're going to output based on the current cell state, but a filtered version. Output from the sigmoid layer gives what parts of the cell state we're going to output. passing the cell state through tanh gives values between -1 and 1 and multiplying this with the output of the sigmoid layer gives the values of parts we only decided to output.

$$o_t = \boldsymbol{\sigma}(W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * tanh(C_t)$$

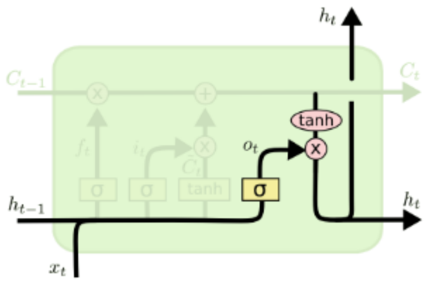


Fig 3.13 Output gate in LSTM cell

3.4.4 LSTM Response to Vanishing/Exploding Gradient Problem

Backpropagation in LSTM is much cleaner compared to traditional RNN. There is no multiplication with the weight matrix during backpropagation. But it is multiplied by different values of the forget gate which makes it less prone to the vanishing/exploding gradient problem.

3.4.5 Network Structure

We decided to use a stacked LSTM for this project, each layer has 50 LSTM cells. Fig 3.14 shows the architecture of the network.

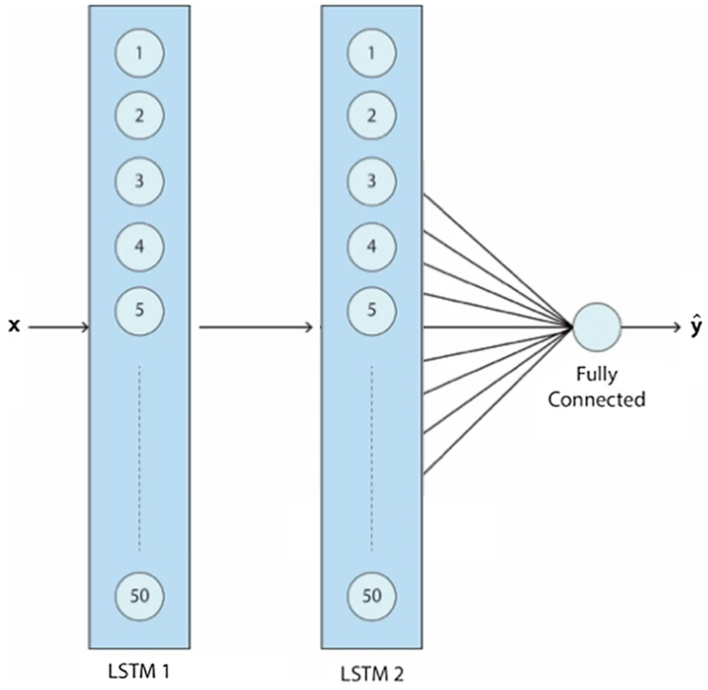
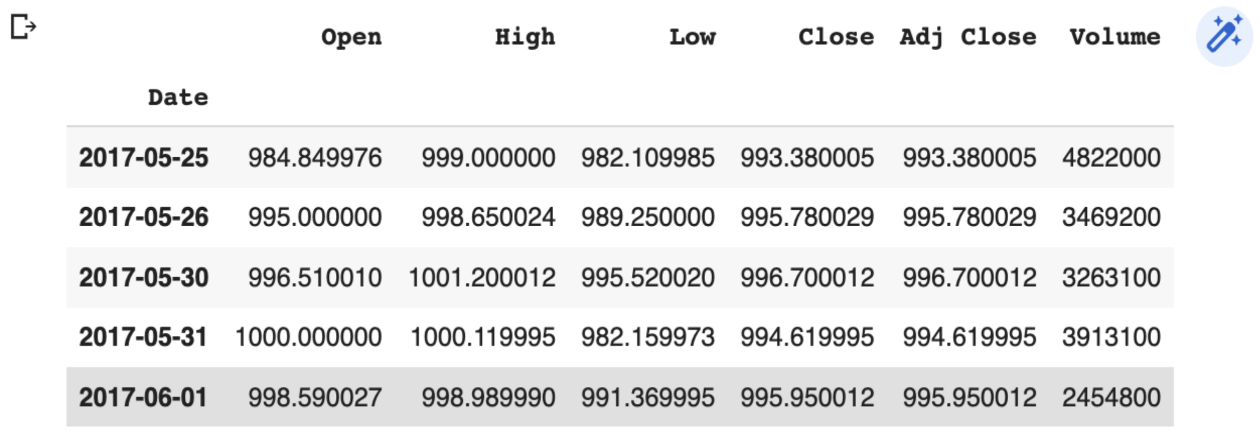


Fig 3.14 Network Structure

4. STEPS TO PREDICT AMAZON STOCK PRICE

Step 1: Downloading the dataset

To begin with, we downloaded Amazon's 5-year's historical data from finance.yahoo.com/. This dataset had 1257 samples starting from May 24, 2017, to May 22, 2022. It had 'Date' as an index and six features namely: 'Open, High, Low, Close, Adj Close, Vol'. Feature 'Open' specifies a stock's opening price for the day, feature 'High' specifies a stock's highest price for the day, feature 'Low' specifies a stock's lowest price for the day, feature 'Close' specifies a stock's closing price for the day, feature 'Adj Close' specifies a stock's adjusted closing price which is a stock's closing price after taking into account any corporate actions and feature 'Vol' specifies the volume of stock for that particular day.



	Open	High	Low	Close	Adj Close	Volume
Date						
2017-05-25	984.849976	999.000000	982.109985	993.380005	993.380005	4822000
2017-05-26	995.000000	998.650024	989.250000	995.780029	995.780029	3469200
2017-05-30	996.510010	1001.200012	995.520020	996.700012	996.700012	3263100
2017-05-31	1000.000000	1000.119995	982.159973	994.619995	994.619995	3913100
2017-06-01	998.590027	998.989990	991.369995	995.950012	995.950012	2454800

Fig 4.1 Amazon Stock Price Dataset

Step 2: Loading Dataset:

We loaded the dataset into our code using 'pandas' data frame and read the '.csv' file with index 'Date' as follows:

```
import pandas as pd
stock_data = pd.read_csv('./AMZN.csv',index_col='Date')
```

Step 3: Data Preprocessing:

First, we chose only those features which are relevant for predicting the future stock price out of the five available features which are: ‘Open, High,Low’. We then chose the target variable to hold values of the feature ‘Close’.

4.3.1 Feature Scaling:

Feature Scaling is an approach used to normalize the range of features in a dataset. When working with real data it is always advisable to normalize or rescale data within a fixed range to ensure that features with large values don't bias the model. We used ‘StandardScalar’ to rescale feature values between -1 and +1.

The following is the logic used for ‘StandardScalar’ which is provided by sklearn in our project:

```
#Feature Scaling
sc = StandardScaler()
X_ft = sc.fit_transform(X_feat.values)
X_ft =
pd.DataFrame(columns=X_feat.columns,data=X_ft,index=X_feat.index)
y_ft = sc.fit_transform(target_y.values)
y_ft =
pd.DataFrame(columns=target_y.columns,data=y_ft,index=target_y.index)
```

Step 4 : Splitting Dataset into Training and Test Sets:

As explained in the architecture, LSTM needs a window of samples in every step of training. For example, if the window size is 10 then LSTM uses 9 samples and assigns weights to them in every training step i.e, it uses 9 samples for training to predict the 10th sample value. We used a method called ‘lstm_split’ which takes preprocessed data from Step 3 and number of steps. This method typically creates windows each of size ‘number of steps’ from the starting of the dataset. The following is the logic used for lstm_split, if number of steps/n_steps = 10 then the first element of list X will have features of first 10 data samples and list y contains target value of 10th data sample.

```
   def lstm_split(data,n_steps):
                X,y=[],[]
for i in range(0,len(data) - n_steps +1):
      X.append(data[i:i + n_steps,:-1])
       y.append(data[i + n_steps-1,-1])

      return np.array(X),np.array(y)
```

We then split the entire data after lstm_split data 80:20 ratio of training and test sets which left us with 999 samples for training and 249 samples for testing out of available 1248 samples.

Step 5 : Implementation of Algorithm:

4.5.1 Simple Recurrent Neural Networks:

Originally Neural Networks are stateless, however Recurrent Neural Networks solve this problem of Neural Networks by using the state of the previous layer and passing the updated information to the next timestamp. Initially, we tried predicting future stock price values of Amazon using Simple Recurrent Neural Networks. We used one input layer with 32 neurons and one output layer. In neural networks, activation functions are used as barriers between input that is being passed to the function and its output. Activation functions decide if a neuron should be activated or not. We used the ‘sigmoid’ function as the activation function. We use a parameter called “returnSequences” and we set this value to “True” since a sequence of outputs should be returned from each layer to be passed on to the next layer.

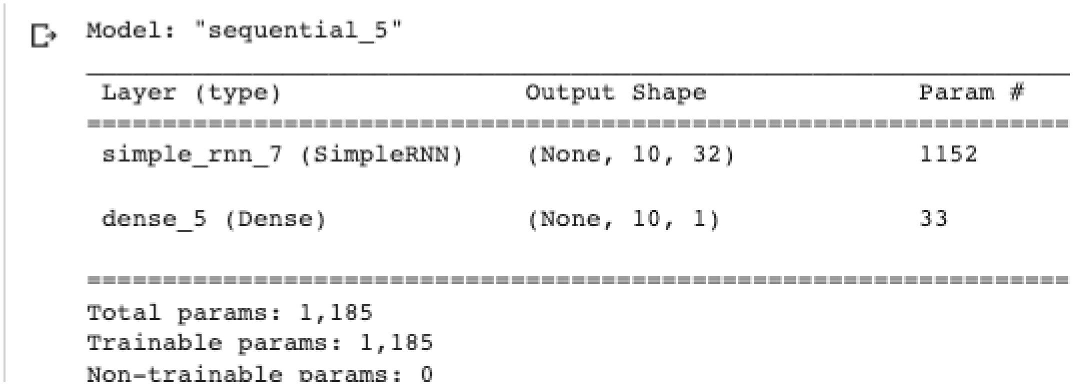
4.5.1.1 Compilation:

Later we compiled the model by specifying the loss function and optimizer. Loss function specifies the difference between the actual and the predicted value. In our project, we used ‘Mean Squared Error’ as our loss function which calculates the mean of squared error between actual and predicted values. Optimizer optimizes the input weights by reducing the loss incurred.

In our project, we used 'Adam' optimizer which is a stochastic gradient descent method and has a faster computation time and needs less number of parameters.

The following piece of code shows the logic for building a Simple RNN Model and it's summary:

```
#building RNN Model
simple_rnn = Sequential()
#input layer
simple_rnn.add(SimpleRNN(32,input_shape=(X_train.shape[1],X_train.sh
ape[2]),activation='sigmoid',return_sequences=True))
#output layer
simple_rnn.add(Dense(1))
simple_rnn.compile(loss='mean_squared_error',optimizer='adam')
simple_rnn.summary()
```



```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn_7 (SimpleRNN)    (None, 10, 32)            1152

 dense_5 (Dense)             (None, 10, 1)             33

=================================================================
Total params: 1,185
Trainable params: 1,185
Non-trainable params: 0
```

Fig 4.2 Simple RNN Summary

The summary specifies the type of the layer, the shape of the layer output, number of parameters of each layer including total parameters and total parameters that are trained.

4.5.1.2 Fitting the model on training data:

After building the model we fit the model on training data with 100 epochs and 4 batches as follows:

```
#fit simple rnn model to training data
```

```
history=
      simple_rnn.fit(X_train,y_train,epochs=100,batch_size=4,verbose=2)
```

4.5.2 Long- Short Term Memory Networks(LSTM):

LSTMs can remember very old information for a long period of time which is an advantage over RNNs. They also tend to solve "Vanishing/Exploding Gradient" problems as discussed in the previous sections. We performed two experiments with LSTMs - one with a single input layer and a single output layer and the other with a single input layer, single hidden layer and a single output layer. We built a 'Sequential' model since all the layers should be placed in a Sequential fashion as data flows from one layer to another in a specified order. We used 'sigmoid' activation function just like the one we used in Simple RNNs.

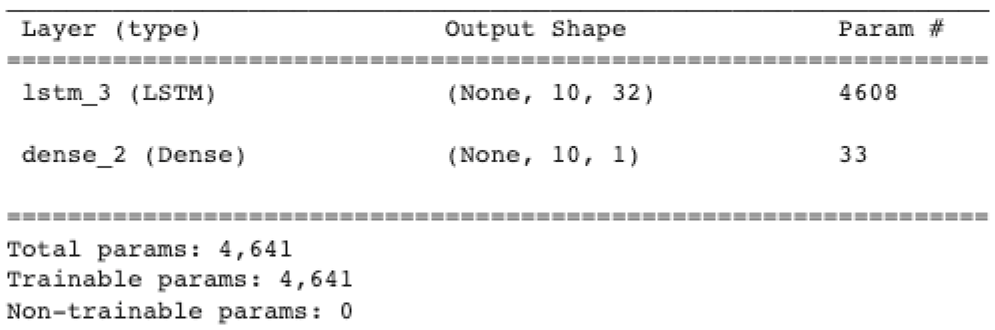
4.5.2.1 Compilation:

We then compiled the model by specifying the Loss function and optimizer. We compiled the LSTM model using the same loss function - "Mean Squared Error" and "Adam" optimizer that we used while compiling the Simple RNN model.

Experiment 1: LSTM Model with a single input layer and a single output layer:

The following shows the code for LSTM model with a single input layer with 32 neurons and a single output layer:

```
#building LSTM Model
lstm = Sequential()
#input layer
lstm.add(LSTM(32,input_shape=(X_train.shape[1],X_train.shape[2]),activ
      ation='sigmoid',return_sequences=True))
#output layer
lstm.add(Dense(1))
lstm.compile(loss='mean_squared_error',optimizer='adam')
lstm.summary()
```



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_3 (LSTM)               (None, 10, 32)            4608

 dense_2 (Dense)             (None, 10, 1)             33

=================================================================
Total params: 4,641
Trainable params: 4,641
Non-trainable params: 0
```

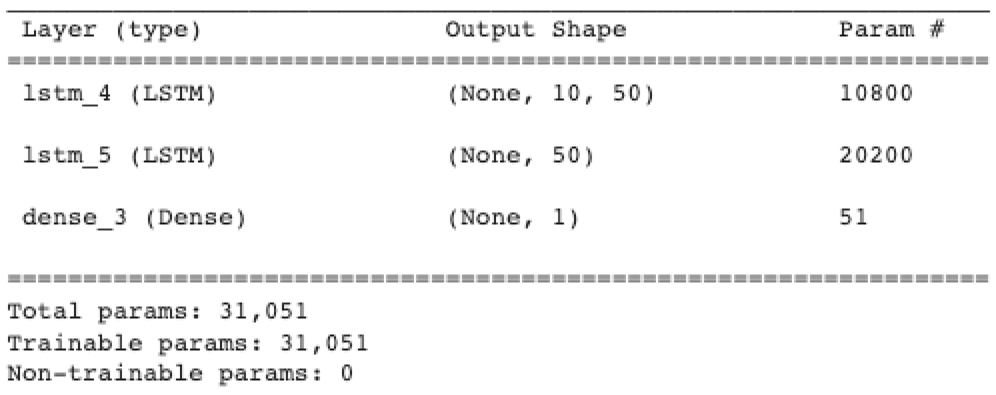
Fig 4.3 LSTM with single input and output layer Summary

The summary specifies the input layer, its shape and its respective number of parameters, and output layer, its shape and its respective number of parameters.

Experiment 2: LSTM Model with a single input layer, single hidden layer and a single output layer:

The following shows the code for LSTM model with a single input layer with 50 neurons, a hidden layer with 50 neurons which uses ‘sigmoid’ activation function and a single output layer:

```
#building LSTM Model
lstm = Sequential()
#input layer
lstm.add(LSTM(50,input_shape=(X_train.shape[1],X_train.shape[2]),activ
      ation='sigmoid',return_sequences=True))
#hidden layer 1
lstm.add(LSTM(50,activation='sigmoid'))
#output layer
lstm.add(Dense(1))
lstm.compile(loss='mean_squared_error',optimizer='adam')
lstm.summary()
```



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_4 (LSTM)                (None, 10, 50)            10800

lstm_5 (LSTM)                (None, 50)                20200

dense_3 (Dense)              (None, 1)                 51

=================================================================
Total params: 31,051
Trainable params: 31,051
Non-trainable params: 0
```

Fig 4.4 LSTM with 1 input layer, 1 hidden layer and 1 output layer

The summary specifies input layer, it’s shape and it’s respective number of parameters;hidden layer, it’s shape and it’s respective number of parameters and output layer,it’s shape and it’s respective number of parameters.

4.5.2.2 Fitting each model on training data:

Later, we fit each model individually on training data with 100 epochs and a batch size of 4 as follows:

```
#fit LSTM model to training data
history = lstm.fit(X_train,y_train,epochs=100,batch_size=4,verbose=2,shuffle=False)
```

Step 6: Predicting the stock prices on test dataset:

Later, we predicted the stock prices on the test dataset using ‘simple_rnn’ model, ‘lstm model with a single input and output layer’ and ‘lstm model with a single input layer, single hidden layer and a single output layer’ as follows:

```
#prediction using simple_rnn model
y_pred = simple_rnn.predict(X_test)


#prediction using lstm model with a single input and output layer
```

```
y_pred =lstm.predict(X_test)

#prediction using lstm model with 1 input layer, 1 hidden layer, 1
     output layer
y_pred =lstm.predict(X_test)
```

On the whole, the time taken to train and predict data was less than 1 min.

5. RESULTS

The following graphs show the difference between actual and predicted values on the test dataset for each model:

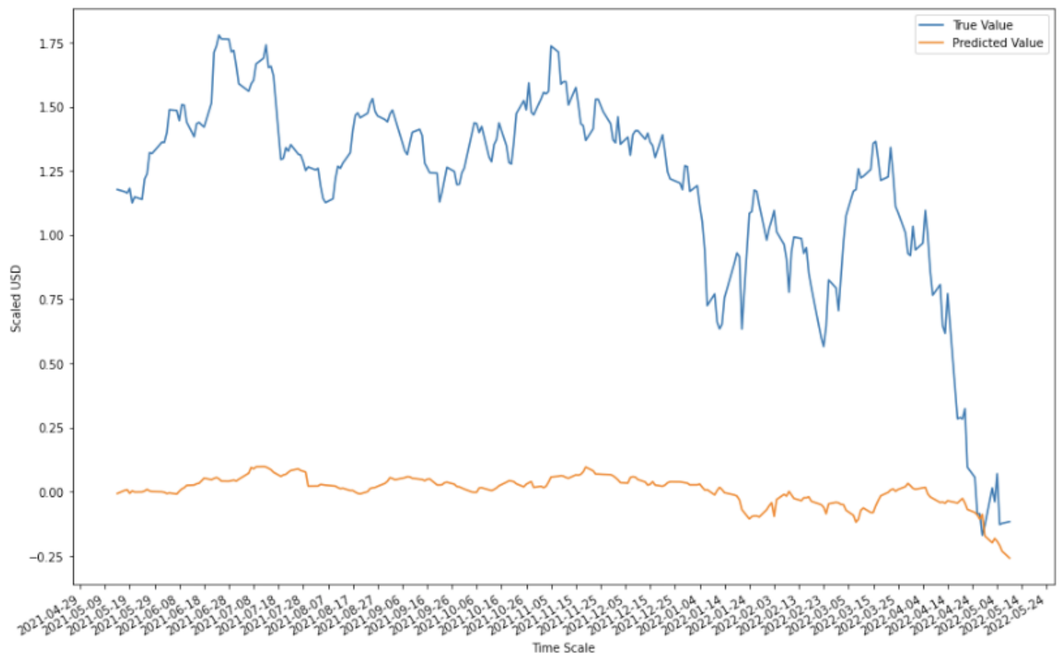


Fig 5.1 Test Results using Simple RNN

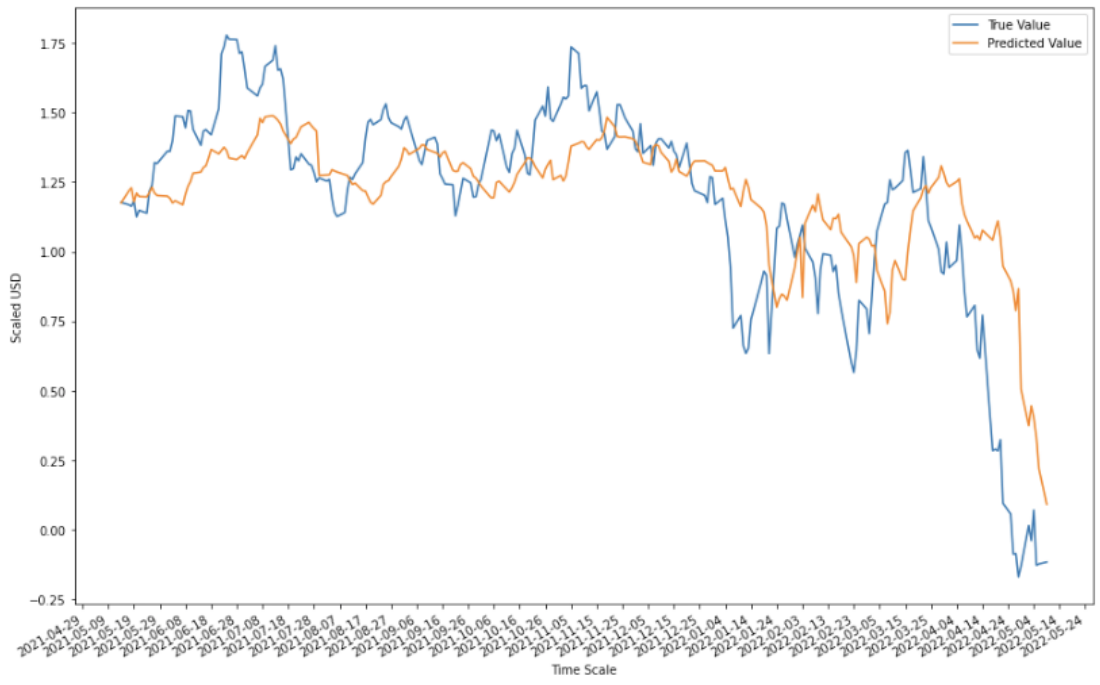


Fig 5.2 Test Results using LSTM with 1 input layer and 1 output layer

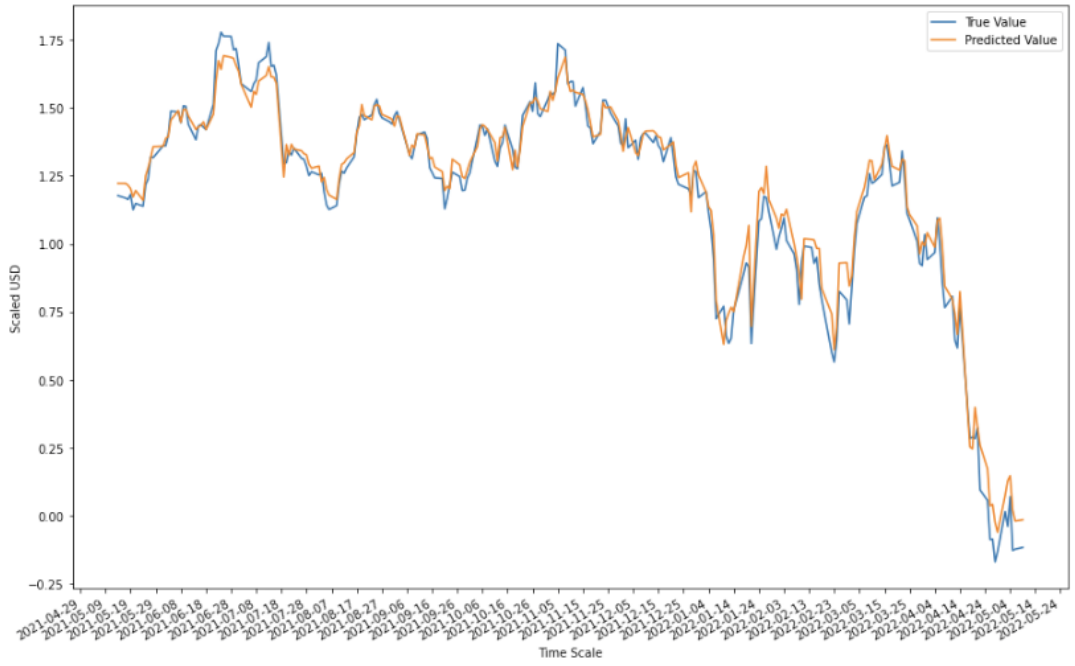


Fig 5.3 Test Results using LSTM with 1 input layer, 1 hidden layer, and 1 output layer

From the above results, we observed that LSTM with 1 input layer, 1 hidden layer, and 1 output layer performed the best when compared with other models.

6. PERFORMANCE METRICS

We used the following performance metrics to evaluate the performance of our test dataset:

6.1 Root Mean Squared Error(RMSE):

RMSE is a commonly used measure to find the difference between actual and predicted values. It calculates the square root of the mean of squared error difference between actual and predicted values.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} \|y(i) - \hat{y}(i)\|^2}{N}},$$

Fig 6.1RMSE

where y(i) = Actual Value , y_hat(i) = Predicted Value, N= number of datapoints

6.2 Mean Absolute Percentage Error(MAPE):

MAPE is another measure which calculates the percentage of mean of absolute difference between actual and predicted values.

$$MAPE = \frac{\sum \frac{|A-F|}{A} \times 100}{N}$$

Fig 6.2 MAPE

where A = Actual Value, F = Predicted value

RMSE and MAPE values for different models are tabularized as follows:

Neural Network Method	RMSE	MAPE
Simple RNN	1.170	0.968
LSTM with 1 input layer,1 output layer	0.275	0.617
LSTM with 1 input layer, 1 hidden layer, 1 output layer	**0.0565**	**0.109**

Table 6.1 RMSE and MAPE values for different models

From the above table, it can clearly be understood that LSTM with 1 input layer, 1 hidden layer, 1 output layer minimized error the most.

7. CONCLUSION & FUTURE SCOPE

In this project, we explored two different Recurrent Neural Networks: Simple Recurrent Neural Networks and Long-Short Term Memory Networks, a type of Recurrent Neural Network. After conducting multiple experiments and monitoring the error between actual and predicted Amazon stock price values on the test dataset, we observed that LSTM with a single input layer, a hidden layer, and an output layer performed better than the other two models.

During the entire project, we got an opportunity to understand Neural Networks particularly Recurrent Neural Networks in-depth and play around with hyperparameter values such as increasing the number of steps and number of units or neurons, to observe better performance.

The scope of this project can be extended by considering posts or tweets describing the nature of a specific stock on social networking sites.

8. REFERENCES

[1] Stock Price Prediction using Machine Learning, April 22, 2022, https://www.projectpro.io/article/stock-price-prediction-using-machine-learning-project/571#mcetoc_1fqoetfs7j (Accessed on April 15, 2022)

[2] Shipra Saxena, March 16, 2021, Introduction to Long Short Term Memory, https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/ (Accessed on April 20, 2022)

[3] Understanding LSTM Networks, August 27, 2015, https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (Accessed on April 12, 2022)