Time Series Forecasting using LSTM: New York Stock Exchange

Athulya Shaji

MSc Computer Science

School of Computing, Engineering and Built-Environment

Ulster University, Belfast
Northern Ireland

Shaji-A@ulster.ac.uk

***Abstract* – In this paper, I am trying to implement the Long Short-Term Memory (LSTM) model, which is a kind of Recurrent Neural Network (RNN) part of Deep Learning under Machine Learning where the efforts are made to forecast the future values of the New York Stock Exchange based on the previous years data.**

I. INTRODUCTION

Artificial Intelligence (AI) is a concept of making computers have intelligence i.e. the ability to think or one can say make decisions on their own. Machine Learning (ML) is a way to make AI possible, where the machine tries to learn on its own from the huge amount of data available with minimum human interference, Deep learning is a method of machine learning, where a neural networking structure is implemented to make decisions from the input data.

Deep learning can implement neural networks in many ways such as Convolutional Neural Networks (CNN) or as Recurrent Neural Networks (RNN) and many more. RNN is, as the name suggests takes recurring inputs to produce output. LSTM is a method of RNN where it while taking recurrent inputs can also hold the past memory which also becomes a factor in determining the current output. Hence LSTM is mainly used on sequential time series data i.e., data that is changing over a period of time.

Here I will be using LSTM which can take multiple time step input and train a model to predict multiple output time steps which will be tested on the available data itself and determine the efficiency of the model and hence will be used to predict the future outputs.

II. DEEP LEARNING

Before starting with the LSTM implementation, it is very important to have background knowledge of deep learning concepts. How deep learning is different from machine learning?

In machine learning feature extraction and classification is done in separate parts, where mostly human interference is needed for feature extraction. Whereas in deep learning feature extraction and classification is done together with less human interference as shown in the below figure.

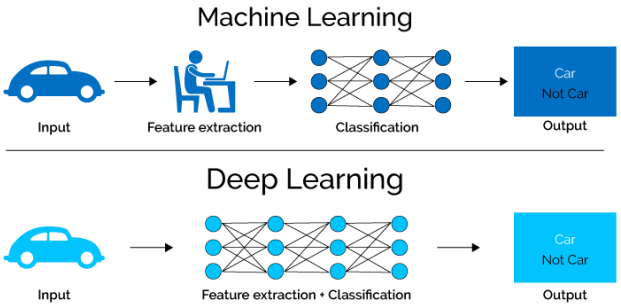


Figure1: Machine learning and deep learning

A *Neural Network* in deep learning is basically trying to replicate a human neuron system. As shown below many input comes into a cell and with the use of an *Activation Function* which determines whether the signal should be passed is determined based on the value from the function These activations functions can be Sigmoid, tanh, ReLu etc

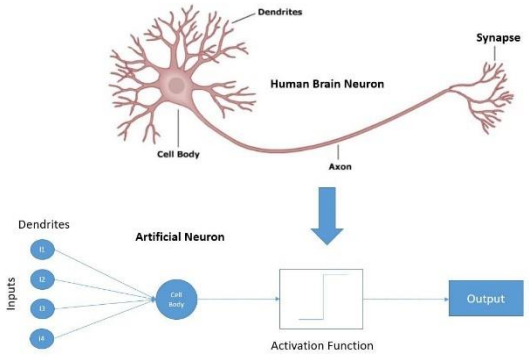


Figure2: A neural network system

III. RNN

When the data we are dealing with is sequential, RNN is the best fit. Traditional methods are more suitable for stationary data, i.e. the data where the mean and standard deviation remains constant throughout the dataset. Whereas datasets such as time-series datasets are non-stationary, meaning the mean of the values are changing as time passes by. In such cases, RNN can be used as it remembers the past, i.e. the input for the next sequential output also contains the previous step's data also.

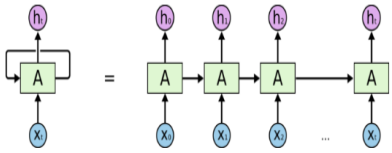


Figure3: RNN

The activation function used by an RNN is tanh.

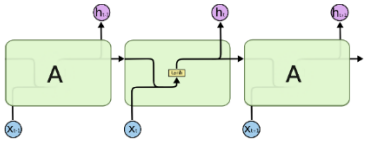


Figure4: An RNN cell

There are different types of RNN:

One to one: where one input yields a single output. Eg: image classification – to determine the object in the image.

One to many: where one input gives much output. Eg: image captioning – to describe an image.

Many to one: many inputs conclude to a single output. Eg: sentiments analysis on social media.

Many to many: multiple inputs giving multiple outputs. Eg: language translation from one language to another.

Limitation of RNN: RNN works best for short sequences. When lots of previous information is needed to determine the next output, RNN fails to perform well, as shown in the below figure.

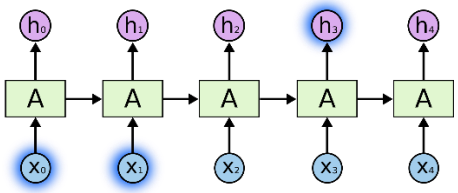


Figure5: RNN fails for Long sequence

IV. LSTM

LSTM is a special kind of RNN, also the most popular type as it is capable to overcome the limitations of RNN discussed in the previous section. LSTM is capable of working with complex tracing and remembering information for a long period of time, making it popular for time series data forecasting and natural language processing.

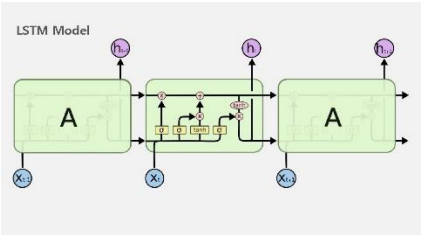


Figure6: A LSTM network

An LSTM cell contains forget gate, input gate, and output gate, which include the Sigmoid function used to forget or remember information (0 – forget, 1- remember) and tanh functions which give weight to cells from -1 to 1. These functions together determine the output using a number of equations.

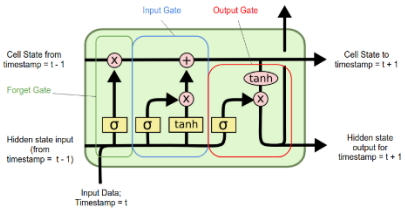


Figure7: An LSTM cell

Gate equations

$$i_t = \sigma(w_i[h_{t-1}, x_t] + b_i)$$
$$f_t = \sigma(w_f[h_{t-1}, x_t] + b_f)$$
$$o_t = \sigma(w_o[h_{t-1}, x_t] + b_o)$$

Cell equation

$$\tilde{c_t} = tanh(w_c[h_{t-1}, x_t] + b_c)$$
$$c_t = f_t * c_{t-1} + i_t * \tilde{c_t}$$
$$h_t = o_t * tanh(c^t)$$

TensorFlow is the most famous library for implementing many deep learning algorithms. LSTM is also inbuild in the library as per the above-mentioned architecture, for implementation of LSTM to create a machine learning

model, we have to invoke the LSTM function from the TensorFlow library and fit it for our dataset.

V. PROBLEM SCENARIO

Here I'm trying to forecast future values from given data using LSTM. The data set available is of the New York Stock Exchange from the year 2000 to 2021. Shown below is the first and last five records of the dataset from all the columns.

Dataset.head():

	Index	Date	Open	High	Low	Close
0	NYA	03-01-2000	6762.109863	6762.109863	6762.109863	6762.109863
1	NYA	04-01-2000	6543.759766	6543.759766	6543.759766	6543.759766
2	NYA	05-01-2000	6567.029785	6567.029785	6567.029785	6567.029785
3	NYA	06-01-2000	6635.439941	6635.439941	6635.439941	6635.439941
4	NYA	07-01-2000	6792.669922	6792.669922	6792.669922	6792.669922

Dataset.tail()

	Index	Date	Open	High	Low	Close
5381	NYA	24-05-2021	16375.00000	16508.51953	16375.00000	16464.68945
5382	NYA	25-05-2021	16464.68945	16525.81055	16375.15039	16390.18945
5383	NYA	26-05-2021	16390.18945	16466.33984	16388.32031	16451.96094
5384	NYA	27-05-2021	16451.96094	16546.35938	16451.96094	16531.94922
5385	NYA	28-05-2021	16531.94922	16588.68945	16531.94922	16555.66016

The intention is to predict 30 future values after the last record from the dataset.

VI. IMPLEMENTATION

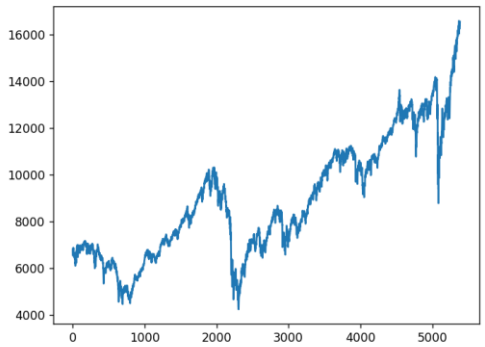
The aforementioned problem has been fulfilled in python using the PyCham IDE.

Following are the libraries used

```
import numpy as np
import matplotlib.pyplot as plt
from pandas import read_csv
import math
import tensorflow
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM, Flatten
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
```

The implementation starts with loading the dataset to the environment using the library pandas, read function. I am only considering the 'Close' column to do the forecasting, hence will be only loading that column and proceeding with exploring the data by cleaning it and doing some visualization of the dataset, and identifying the characteristic of the dataset like seasonality, p_value, correlation, etc.

```
# load the dataset
df = read_csv('C:/Users/DELL/Downloads/NYA_Stock1.csv', usecols=[5])

#clean dataset
df = df[df['Close'].notna()]
df=df.reset_index()['Close']
print(df.head())
print(df.tail())
df = df.values
df = df.astype('float32')

#plotting dataset
plt.plot(df)
plt.show()
```



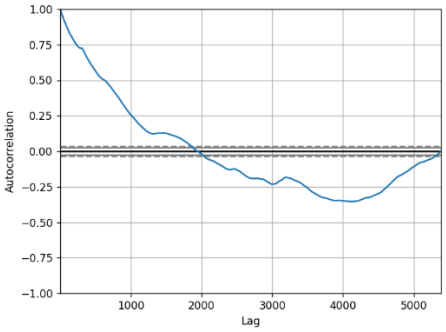
The p values determine whether the data is stationary or non-stationary.

Output:

'p-value = 0.9665320212538789 if above 0.05, data is not stationary'

Since time-series datasets are correlated with their own previous values, it is best to plot an autocorrelation to identify the optimum time step

```
#plotting autocorrelation
from pandas.plotting import autocorrelation_plot
autocorrelation_plot(df)
plt.show()
```



From the output from the autocorrelation, it's clear that values below 500 will be best, hence I have taken time steps as 100. i.e. the first 100 values will be used as input for getting the 101st value as output.

LSTM uses sigmoid and tanh that are sensitive to magnitude, so values need to be normalized. Also, for splitting data into train and test, random values can't be taken since it is a time series function, hence first 65% is taken for training and the remaining for testing.

```
# normalize the dataset
scaler=MinMaxScaler(feature_range=(0,1))
df=scaler.fit_transform(np.array(df).reshape(-1,1))

# split into train and test sets
training_size=int(len(df)*0.65)
test_size=len(df)-training_size
train_data,test_data=df[0:training_size,:],df[training_size:len(df),:1]
```

Finally, to fit LSTM models, we have to convert the data into a matrix of X and Y where X is an array of input arrays divided into the number of time steps and Y consist of the corresponding output values. The X and Y train are reshaped into 3D array to fit into LSTM parameters.

```
#create x_train,y_train,x_test,y_test
def create_dataset(dataset, time_step=1):
    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)

# Number of time steps to look back
time_step = 100
x_train, y_train = create_dataset(train_data, time_step)
x_test, y_test = create_dataset(test_data, time_step)

# reshape input to be [samples, time steps, features] which is required for LSTM
x_train =x_train.reshape(x_train.shape[0],x_train.shape[1] , 1)
x_test = x_test.reshape(x_test.shape[0],x_test.shape[1] , 1)
```

The LSTM model is created by 3 stacks of LSTM and 1 dense layer. And fitted using the above arrays (X and Y) with 50 epochs. Once the model is created the model is used to predict the output for the train and test data.

```
# Stacked LSTM model creation
model=Sequential()
model.add(LSTM(50,return_sequences=True,input_shape=(100,1)))
model.add(LSTM(50,return_sequences=True))
model.add(LSTM(50))
model.add(Dense(1))
model.compile(loss='mean_squared_error',optimizer='adam')
model.summary()
model.fit(x_train,y_train,validation_data=(x_test,y_test),epochs=50,batch_size=64,verbose=1

# make predictions
train_predict=model.predict(x_train)
test_predict=model.predict(x_test)
```

The data arrays which were scaled for fitting in the model are transformed back to original values and a performance metric is calculated

```
#Transformback to original form
train_predict=scaler.inverse_transform(train_predict)
test_predict=scaler.inverse_transform(test_predict)
y_train =y_train.reshape(y_train.shape[0],1)
y_test =y_test.reshape(y_test.shape[0],1)
y_train=scaler.inverse_transform(y_train)
y_test=scaler.inverse_transform(y_test)

#performance metrics
print('Performance Metrics')
print('RMSE',math.sqrt(mean_squared_error(y_train[:,0],train_predict[:,0])))
### Test Data RMSE
print('RMSE:',math.sqrt(mean_squared_error(y_test[:,0],test_predict[:,0])))
```

Output:

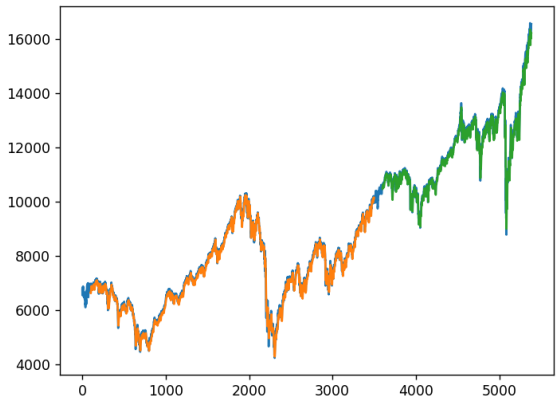
Performance Metrics

RMSE 103.98327502235637

RMSE: 148.5633413943359

The predicted values are plotted against the actual values to check the sync.

```
# Shifting value and Plotting

look_back=100
train_predict1 = np.empty_like(df)
train_predict1[:, :] = np.nan
train_predict1[look_back:len(train_predict)+look_back, :] = train_predict

test_predict1 = np.empty_like(df)
test_predict1[:, :] = np.nan
test_predict1[len(train_predict)+(look_back*2)+1:len(df)-1, :] = test_predict

plt.plot(scaler.inverse_transform(df))
plt.plot(train_predict1)
plt.plot(test_predict1)
plt.show()
```

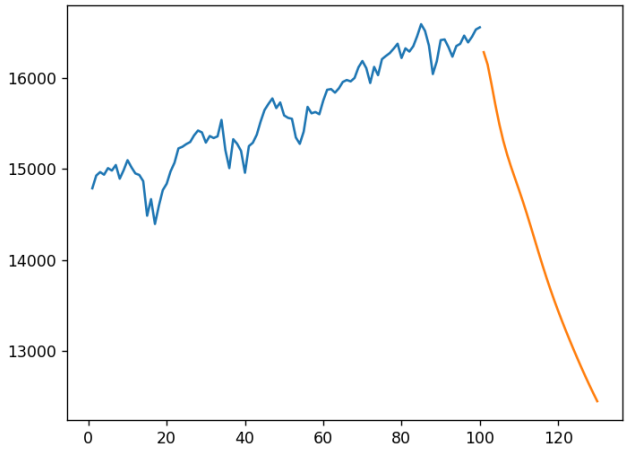


From the graph, the values are greatly coinciding making it a good model for future prediction.

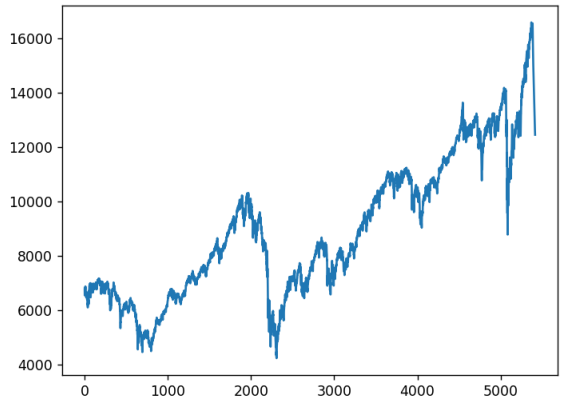
```
# predicting next 30 future stock value
k=len(test_data) - 100
x_input=test_data[k:].reshape(1,-1)
x_input.shape
temp_input=list(x_input)
temp_input=temp_input[0].tolist()
y_output = []
n_steps = 100
i = 0
while (i < 30):

    if (len(temp_input) > 100):
        x_input = np.array(temp_input[1:])
        x_input = x_input.reshape(1, -1)
        x_input = x_input.reshape((1, n_steps, 1))
        # print(x_input)
        y_out = model.predict(x_input, verbose=0)
        temp_input.extend(y_out[0].tolist())
        temp_input = temp_input[1:]
        y_output.extend(y_out.tolist())
        i = i + 1
    else:
        x_input = x_input.reshape((1, n_steps, 1))
        y_out = model.predict(x_input, verbose=0)
        temp_input.extend(y_out[0].tolist())
        y_output.extend(y_out.tolist())
        i = i + 1
print('Future 30 stock value predicted :')
```

To predict the future values, the last 100 values are taken as the input i.e., the dataset had 5385 data rows, to predict the 5386th value, I had to take values from the 5286th index. After finding the first future value, it is added to the existing data, and now the latest 100 values will be used to determine the next value. A similar process is done for finding 30 future values. The forecasted value is being plotted along with the last 100 values of the data as shown in the below graph.

```
day_new=np.arange(1,101)
day_pred=np.arange(101,131)
k=len(df)-100
plt.plot(day_new,scaler.inverse_transform(df[k:]))
plt.plot(day_pred,scaler.inverse_transform(y_output))
plt.show()
df3=df.tolist()
df3.extend(y_output)
df3=scaler.inverse_transform(df3).tolist()
plt.plot(df3)
plt.show()
```



Finally, the entire data and the forecasted data are plotted together



VIII. Conclusion

I had tried to implement the Long Short-Term Memory (LSTM) model to forecast the future values of the New York Stock Exchange based on the previous years' data. The model came out to be good when plotted on top of the actual data. Hence the model was used to predict the next 30 values and had successfully joined with existing data and plotted the graph. The model can be improved by optimizing various parameters like time-steps, and epochs (Too high epochs may lead to overfitting, making the model inaccurate to other datasets), changing the activation function used inside the LSTM, and many more to develop the best model.

References

[1] 70 - An overview of deep learning and neural networks. (2019, November 18).

[2] Cody. (n.d.). Stock Exchange Data [Data set].

[3] (N.d.-a). Licdn.Com. Retrieved May 6, 2022, from https://media-exp1.licdn.com/dms/image/C4E12AQHKRLh_iq3okA/article-cover_image-shrink_423_752/0/1580296292841?e=1657152000&v=beta&t=eI2CWD4k0dw_yg4SWFro1iH7TvRJpbxPGt1Pg_OsKjs

[4](N.d.-b). Deepai.Org. Retrieved May 6, 2022, from https://images.deepai.org/glossary-terms/cf6448554ec74f7fb58cf83192b6e904/synapse.jpeg

[5](N.d.-c). Medium.Com. Retrieved May 6, 2022, from https://miro.medium.com/max/627/1*go8PHsPNbbV6qRiwpUQ5BQ.png

[6](N.d.-d). Codeproject.Com. Retrieved May 6, 2022, from https://www.codeproject.com/KB/AI/1272354/rnn_chain.png

[7](N.d.-e). Github.Io. Retrieved May 6, 2022, from https://colah.github.io/posts/2015-08-Understanding-LSTMs/img/RNN-shorttermdepdencies.png

[8](N.d.-f). Slidetodoc.Com. Retrieved May 6, 2022, from https://slidetodoc.com/presentation_image_h/8f442e350255fcd7142f3b944e7ecd46/image-19.jpg

[9](N.d.-g). Medium.Com. Retrieved May 6, 2022, from https://miro.medium.com/max/1400/1*ahafyNt0Ph_J6Ed9_2hvdg.png