INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

Stock Market Prediction

Author : Deeksha Rawat
Course Project : Fundamentals of Inferential Statistics and Automation(EE798Q)
Course Instructor : Prof. Tushar Sandhan

July 8, 2023

1 LSTM AND ITS VARIOUS STRUCTURES

LSTM stands for **Long Short-Term Memory**. It is a type of **recurrent neural network (RNN)** architecture that is designed to address the vanishing gradient problem, which is a common challenge in traditional RNNs. LSTMs are particularly effective in processing and making predictions based on sequential data, such as time series, text, speech, and audio.

The key feature of LSTMs is their ability to capture and remember long-term dependencies in the input data. This is achieved through a memory cell, which is responsible for storing and updating information over time. The memory cell consists of three main components: **an input gate, a forget gate, and an output gate**.

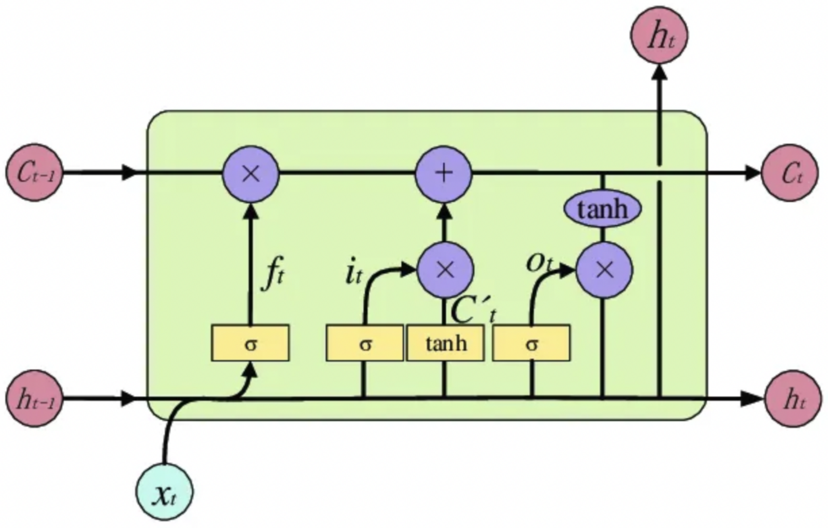
The input gate determines how much new information should be stored in the memory cell, while the forget gate controls the retention or removal of information from the memory cell. The output gate determines how much of the memory cell's content should be used to generate the output at each time step.

The use of gates and the memory cell structure enable LSTMs to handle long sequences of data and preserve relevant information over extended periods, addressing the vanishing gradient problem. This makes LSTMs well-suited for tasks that require modeling dependencies and capturing context from sequential data.

LSTMs have been successfully applied in various fields, including natural language processing, speech recognition, machine translation, sentiment analysis, and time series forecasting. Their ability to process and model sequential data has made them a popular choice in many applications that involve sequential information.

1.1 Classic LSTM

This architecture consists of 4 gating layers through which the cell state works, i.e., **2-input gates, forget gate** and **output gates**.
The input gates work together to choose the input to add to the cell state.
The forget gate decides what old cell state to forget based on current cell state.
The output gates decides what output to be sent through them.



Equations to different gates can be written as:
Forget Gate:

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

Input Gate:

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

$$C'_t = \tanh(W_c.[h_{t-1}, x_t] + b_c)$$

Updated Equation:

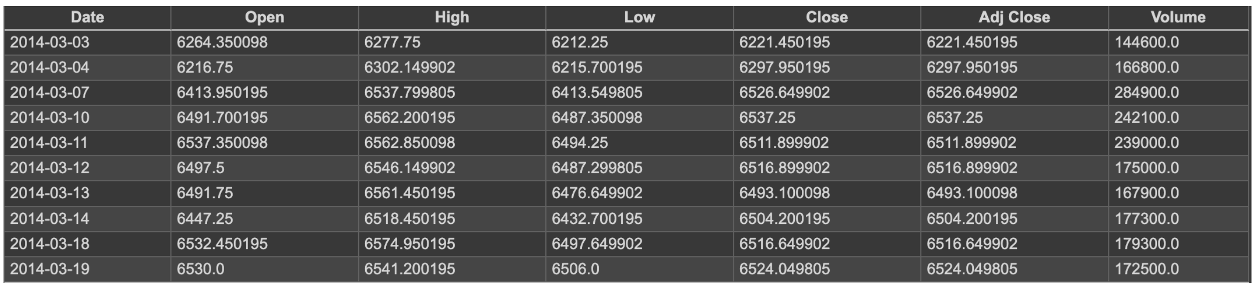
$$C_t = f_t * C_{t-1} + i_t * C'_t$$

Output Gate and Hidden State:

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

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

We have to apply the LSTM model on a 50 days data to predict the closing price of the next two days.

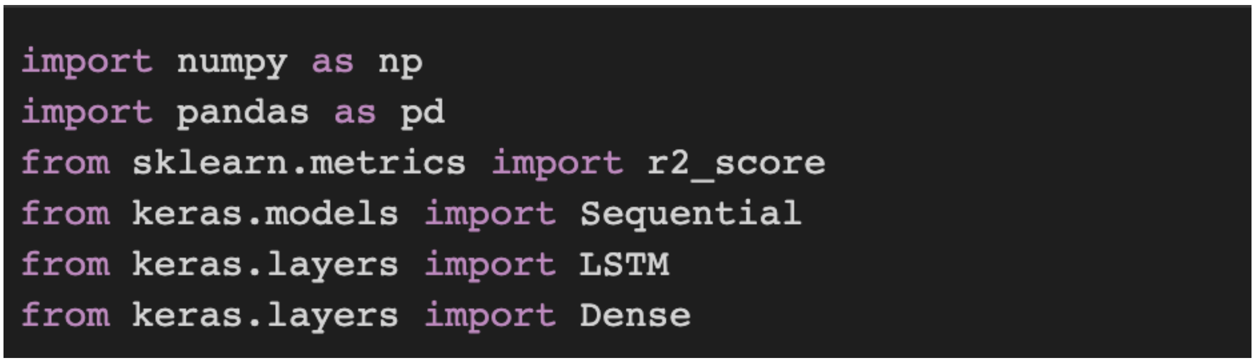


Date	Open	High	Low	Close	Adj Close	Volume
2014-03-03	6264.350098	6277.75	6212.25	6221.450195	6221.450195	144600.0
2014-03-04	6216.75	6302.149902	6215.700195	6297.950195	6297.950195	166800.0
2014-03-07	6413.950195	6537.799805	6413.549805	6526.649902	6526.649902	284900.0
2014-03-10	6491.700195	6562.200195	6487.350098	6537.25	6537.25	242100.0
2014-03-11	6537.350098	6562.850098	6494.25	6511.899902	6511.899902	239000.0
2014-03-12	6497.5	6546.149902	6487.299805	6516.899902	6516.899902	175000.0
2014-03-13	6491.75	6561.450195	6476.649902	6493.100098	6493.100098	167900.0
2014-03-14	6447.25	6518.450195	6432.700195	6504.200195	6504.200195	177300.0
2014-03-18	6532.450195	6574.950195	6497.649902	6516.649902	6516.649902	179300.0
2014-03-19	6530.0	6541.200195	6506.0	6524.049805	6524.049805	172500.0

2 Different LSTM Models

2.1 Vanilla LSTM

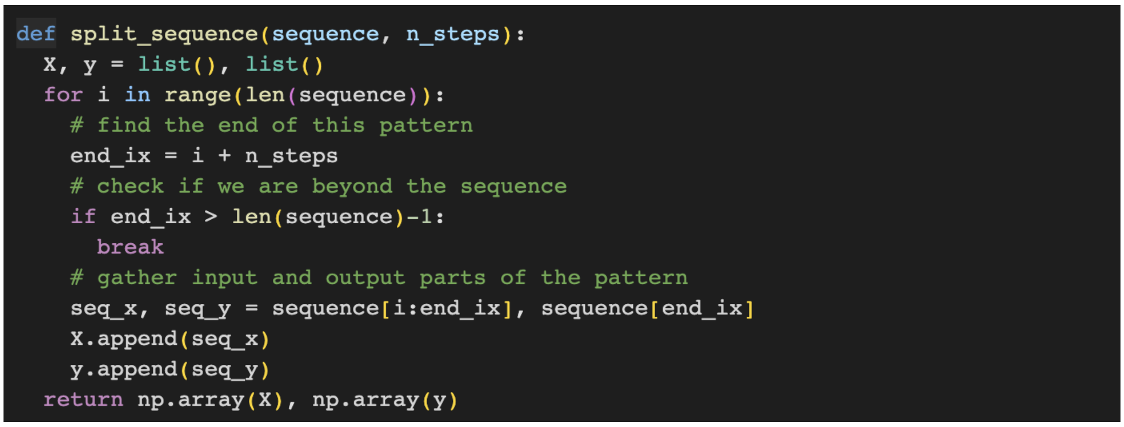
A Vanilla LSTM is an LSTM model that has a single hidden layer of LSTM units, and an output layer used to make a prediction.
We can define a Vanilla LSTM for univariate time series forecasting as follows:



```
import numpy as np
import pandas as pd
from sklearn.metrics import r2_score
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
```

Figure 2.1: Importing the libraries

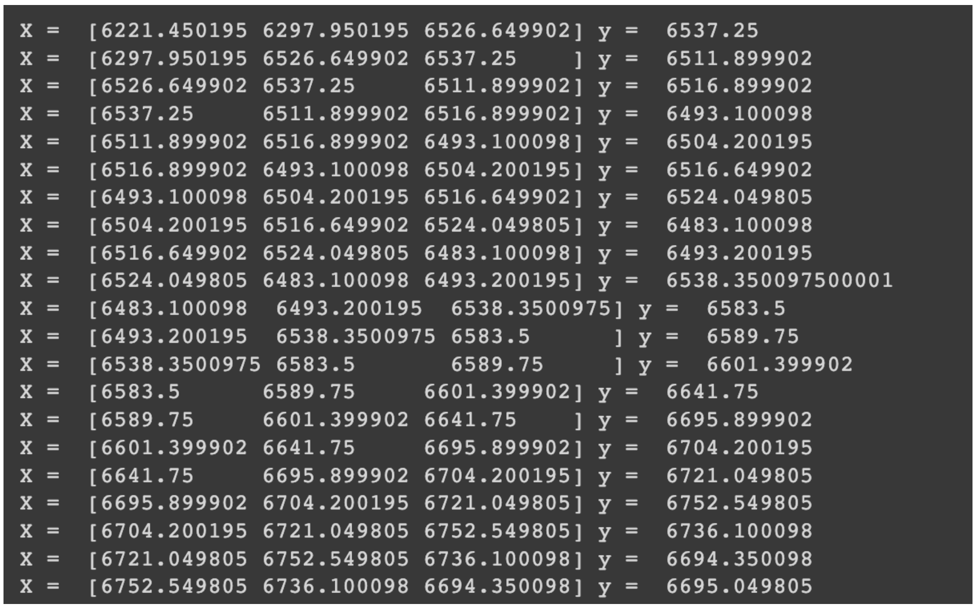
We can divide the sequence into multiple input/output patterns called samples, where three time steps are used as input and one time step is used as output for the one-step prediction that is being learned.



```python
def split_sequence(sequence, n_steps):
  X, y = list(), list()
  for i in range(len(sequence)):
    # find the end of this pattern
    end_ix = i + n_steps
    # check if we are beyond the sequence
    if end_ix > len(sequence)-1:
      break
    # gather input and output parts of the pattern
    seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
    X.append(seq_x)
    y.append(seq_y)
  return np.array(X), np.array(y)
```

Figure 2.2: Splitting the sequence

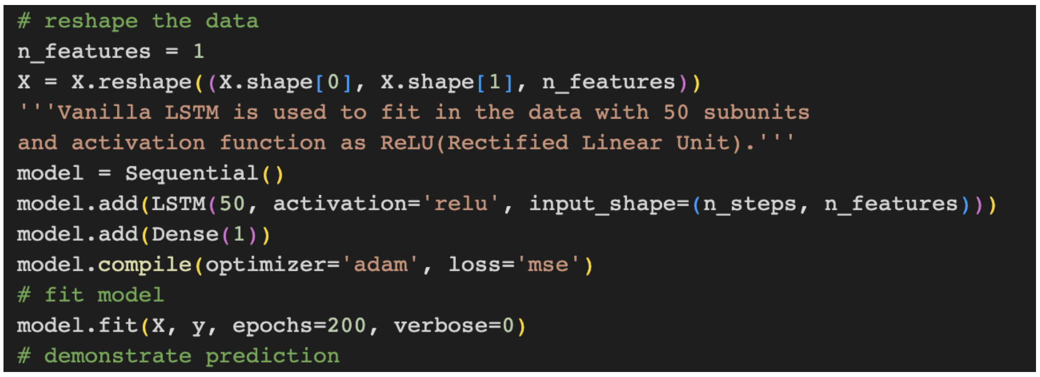
The *split_sequence*() function will split a given univariate sequence into multiple samples where each sample has a specified number of time steps and the output is a single time step.



```
X =  [6221.450195 6297.950195 6526.649902] y =  6537.25
X =  [6297.950195 6526.649902 6537.25    ] y =  6511.899902
X =  [6526.649902 6537.25     6511.899902] y =  6516.899902
X =  [6537.25     6511.899902 6516.899902] y =  6493.100098
X =  [6511.899902 6516.899902 6493.100098] y =  6504.200195
X =  [6516.899902 6493.100098 6504.200195] y =  6516.649902
X =  [6493.100098 6504.200195 6516.649902] y =  6524.049805
X =  [6504.200195 6516.649902 6524.049805] y =  6483.100098
X =  [6516.649902 6524.049805 6483.100098] y =  6493.200195
X =  [6524.049805 6483.100098 6493.200195] y =  6538.350097500001
X =  [6483.100098  6493.200195  6538.3500975] y =  6583.5
X =  [6493.200195  6538.3500975 6583.5      ] y =  6589.75
X =  [6538.3500975 6583.5       6589.75    ] y =  6601.399902
X =  [6583.5      6589.75     6601.399902] y =  6641.75
X =  [6589.75     6601.399902 6641.75    ] y =  6695.899902
X =  [6601.399902 6641.75     6695.899902] y =  6704.200195
X =  [6641.75     6695.899902 6704.200195] y =  6721.049805
X =  [6695.899902 6704.200195 6721.049805] y =  6752.549805
X =  [6704.200195 6721.049805 6752.549805] y =  6736.100098
X =  [6721.049805 6752.549805 6736.100098] y =  6694.350098
X =  [6752.549805 6736.100098 6694.350098] y =  6695.049805
```

Figure 2.3: Result of Split Sequence

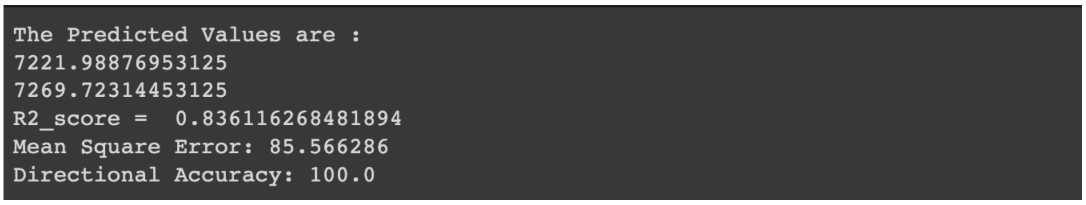
In this case, we define a model with 50 LSTM units in the hidden layer and an output layer that predicts a single numerical value. The model is fit using the efficient Adam version of stochastic gradient descent and optimized using the mean squared error, or 'MSE' loss function. Once the model is defined, we can fit it on the training dataset.



```python
# reshape the data
n_features = 1
X = X.reshape((X.shape[0], X.shape[1], n_features))
'''Vanilla LSTM is used to fit in the data with 50 subunits
and activation function as ReLU(Rectified Linear Unit).'''
model = Sequential()
model.add(LSTM(50, activation='relu', input_shape=(n_steps, n_features)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# fit model
model.fit(X, y, epochs=200, verbose=0)
# demonstrate prediction
```

Figure 2.4: Code for Implementation

The Actual values are : **7232.200195, 7277.899902**
Our model predicted the next two days values as : **7221.98876953125, 7269.72314453125**



```
The Predicted Values are :
7221.98876953125
7269.72314453125
R2_score =  0.836116268481894
Mean Square Error: 85.566286
Directional Accuracy: 100.0
```

Figure 2.5: Results

The **R2 Score** of the Vanilla LSTM model is **0.8361** with a **mean squared error** of **85.5663** and **Directional Accuracy** of **100 percent**.

2.2 Stacked LSTM

Multiple hidden LSTM layers can be stacked one on top of another in what is referred to as a Stacked LSTM model. An LSTM layer requires a three-dimensional input and LSTMs by default will produce a two-dimensional output as an interpretation from the end of the sequence. We can address this by having the LSTM output a value for each time step in the input data by setting the $return\_sequences = True$ argument on the layer. This allows us to have 3D output from hidden LSTM layer as input to the next.

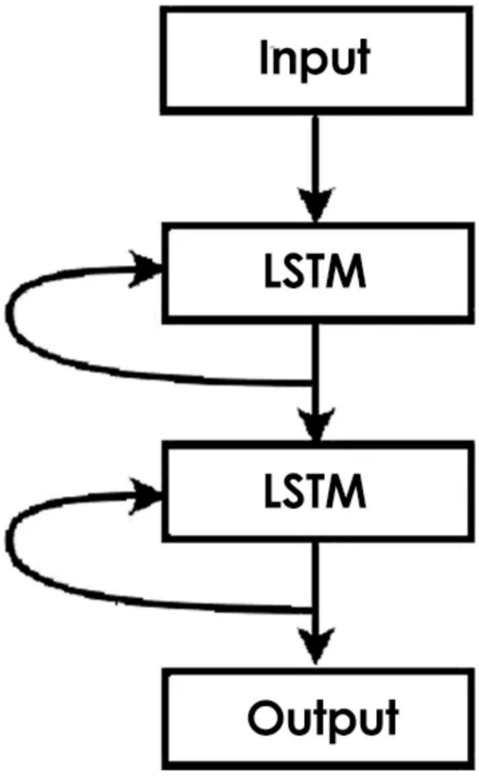
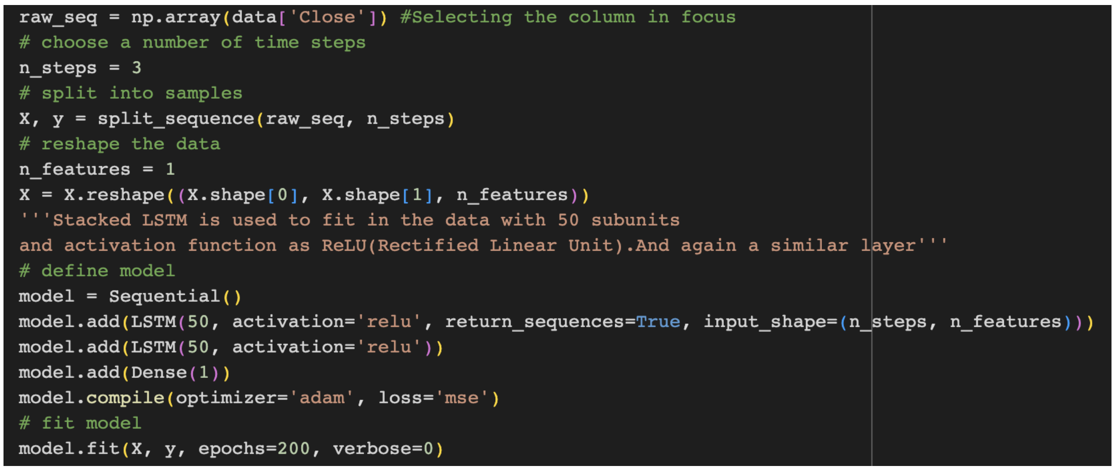


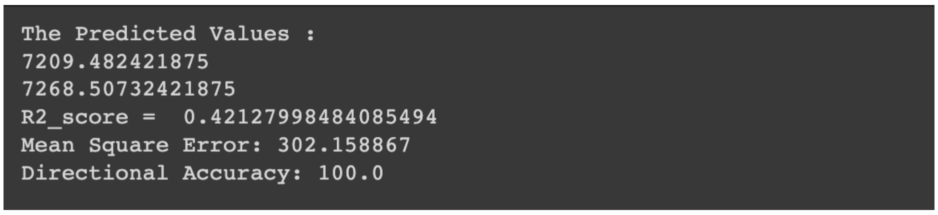
Figure 2.6: Structure of Stacked LSTM



```
raw_seq = np.array(data['Close']) #Selecting the column in focus
# choose a number of time steps
n_steps = 3
# split into samples
X, y = split_sequence(raw_seq, n_steps)
# reshape the data
n_features = 1
X = X.reshape((X.shape[0], X.shape[1], n_features))
'''Stacked LSTM is used to fit in the data with 50 subunits
and activation function as ReLU(Rectified Linear Unit).And again a similar layer'''
# define model
model = Sequential()
model.add(LSTM(50, activation='relu', return_sequences=True, input_shape=(n_steps, n_features)))
model.add(LSTM(50, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# fit model
model.fit(X, y, epochs=200, verbose=0)
```

Figure 2.7: Code for Implementation

This model predicted the next two days values as : **7209.482421875, 7268.50732421875**



```
The Predicted Values :
7209.482421875
7268.50732421875
R2_score =  0.42127998484085494
Mean Square Error: 302.158867
Directional Accuracy: 100.0
```

Figure 2.8: Results

The **R2 Score** of the Stacked LSTM model is **0.4213** with a **mean squared error** of **302.1589** and **Directional Accuracy** of **100 percent**.

2.3 Bidirectional LSTM

It can be beneficial to allow the LSTM model to learn the input sequence **both forward and backwards and concatenate both interpretations**. This is called a Bidirectional LSTM. We can implement a Bidirectional LSTM for univariate time series forecasting by wrapping the first hidden layer in a **wrapper layer** called **Bidirectional**.

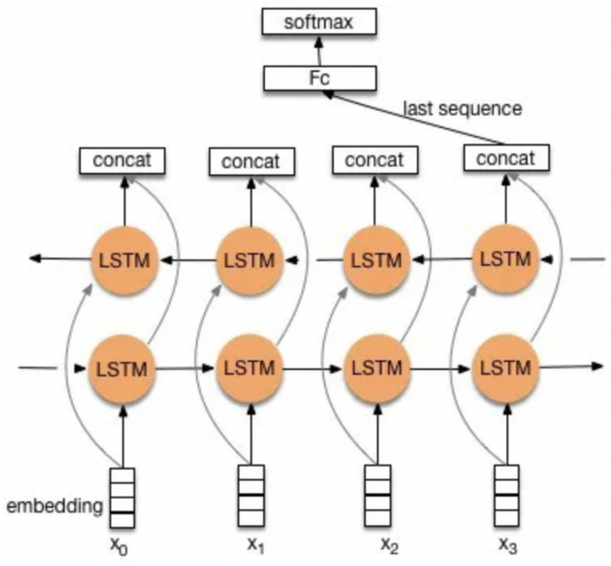
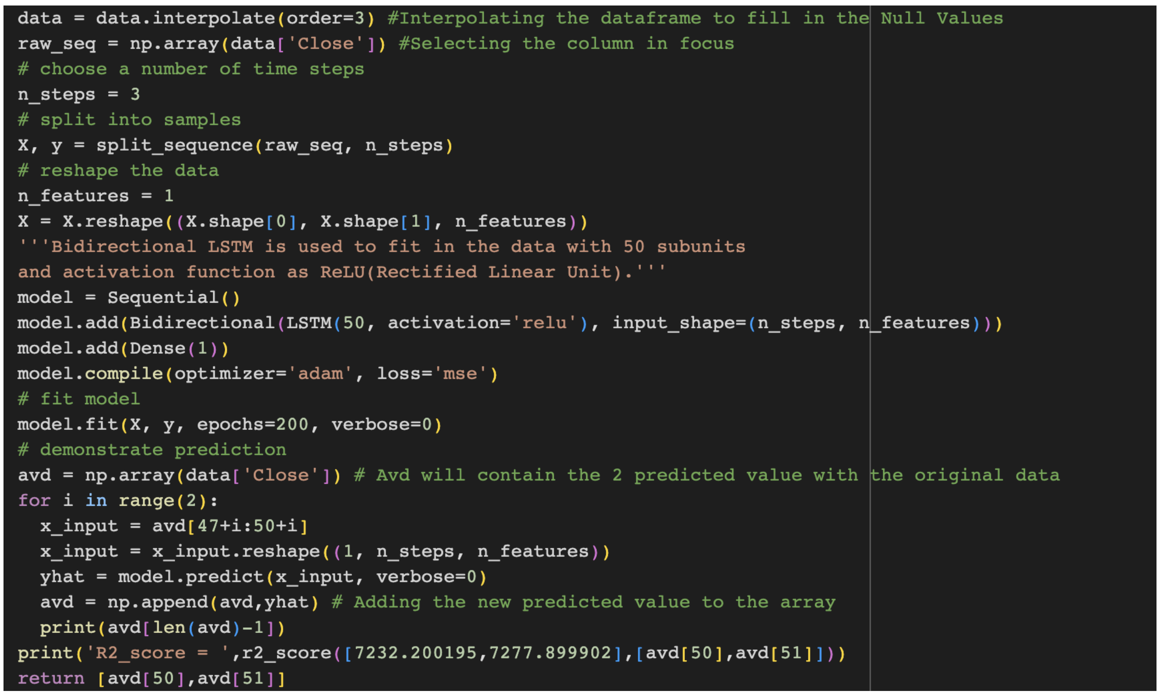


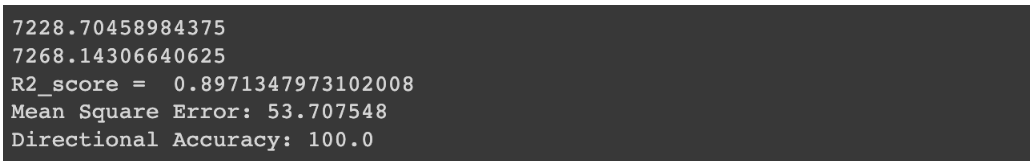
Figure 2.9: Structure of Bidirectional LSTM



```
data = data.interpolate(order=3) #Interpolating the dataframe to fill in the Null Values
raw_seq = np.array(data['Close']) #Selecting the column in focus
# choose a number of time steps
n_steps = 3
# split into samples
X, y = split_sequence(raw_seq, n_steps)
# reshape the data
n_features = 1
X = X.reshape((X.shape[0], X.shape[1], n_features))
'''Bidirectional LSTM is used to fit in the data with 50 subunits
and activation function as ReLU(Rectified Linear Unit).'''
model = Sequential()
model.add(Bidirectional(LSTM(50, activation='relu'), input_shape=(n_steps, n_features)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# fit model
model.fit(X, y, epochs=200, verbose=0)
# demonstrate prediction
avd = np.array(data['Close']) # Avd will contain the 2 predicted value with the original data
for i in range(2):
  x_input = avd[47+i:50+i]
  x_input = x_input.reshape((1, n_steps, n_features))
  yhat = model.predict(x_input, verbose=0)
  avd = np.append(avd,yhat) # Adding the new predicted value to the array
  print(avd[len(avd)-1])
print('R2_score = ',r2_score([7232.200195,7277.899902],[avd[50],avd[51]]))
return [avd[50],avd[51]]
```

Figure 2.10: Code for Implementation

This model predicted the next two days values as : **7228.70458984375, 7268.14306640625**



```
7228.70458984375
7268.14306640625
R2_score =  0.8971347973102008
Mean Square Error: 53.707548
Directional Accuracy: 100.0
```

Figure 2.11: Result

The **R2 Score** of the BiDirectional LSTM model is **0.8971** with a **mean squared error** of **53.7075** and **Directional Accuracy** of **100 percent**.
In our case the best results are given by the **Bidirectional LSTM** with the least MSE.

3 References

- Machine Learning Mastery
- Medium.com