University of Science and Technology of Hanoi

Machine Learning and Data Mining

Data mining for weather prediction and climate change studies

---

Do Dang Minh Duc: USTHBI8-042

Vu Tuan Phong: USTHBI8-139

Tran Kim Quoc Tuan: USTHBI8-159

Phan Manh Tung: USTHBI8-160

Ngo Thanh Tung: USTHBI8-163

April 21, 2020

Contents

1 Introduction **2**
1.1 What is this project about? . . . 2
1.1.1 Dataset . . . 2
1.1.2 Task . . . 2
1.2 Why do we need this project? . . . 2

2 Neural Network **3**
2.1 What is a Neural Network? . . . 3
2.2 The main stages of a Neural Network . . . 3

3 Recurrent Neural Network (RNN) **5**
3.1 What's it about? . . . 5
3.2 Short-term memory and vanishing gradient problem . . . 5

4 Long Short Term Memory (LSTM) **7**

5 Weather Forecasting Implementation **8**
5.1 Preprocess Data . . . 8
5.2 Build LSTM model . . . 11
5.3 Testing and Evaluation . . . 11

Introduction

1.1 What is this project about?

In this report, we would present our weather forecasting project using Long Short Term Memory (LSTM).

The programming language is Python 3. For the platform, we decided to choose Google Colab for strong computing power and convenience of sharing code among members and others.

1.1.1 Dataset

Dataset Link. The dataset consists of 360 data points, recording the temperature hourly.

1.1.2 Task

Task: The problem is time-series forecasting, in which there is only one feature in the data recorded through time.

Therefore, the choice of other regression models which requires at least 2 features might not properly work in this case. LSTM is our best solution proposed for this problem.

1.2 Why do we need this project?

Weather prediction is very important in our daily life. We use the information from weather prediction to plan outdoor activities. A good case in point is predicting the rainy day for planning sporting activities. Secondly, it helps people avoid weather-related health problems like heat stresses, asthma or allergies.

Last but not least, it is also important for the farmers to plan for the harvest. For instance, if the weather is too hot, the farmers should nurture the plants with more water.

Neural Network

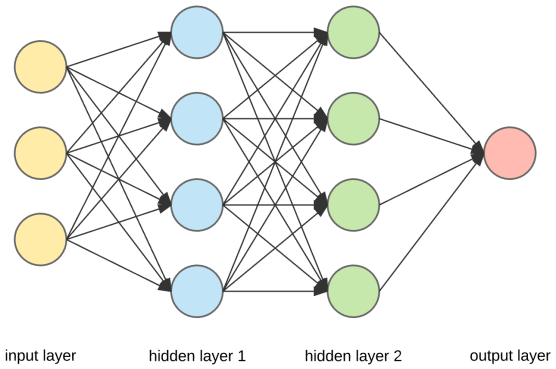
2.1 What is a Neural Network?

A network consists of an input and output layer, with hidden layers in-between.

Made up of layers of neurons/nodes. Neurons of one layer connected to neurons of the next layer through what's called 'Channels'. Each of these channels is assigned a numerical value, known as weight. The hidden layers are responsible for most of the computation before finally coming to an end.

The neural network takes in data as input and uses them to predict the final output.

There are three main stages: Feed the input Train Predict the output.



2.2 The main stages of a Neural Network

The input layer receives inputs. Each input multiply with the corresponding weight and their sum is sent as input to the neurons in the hidden layers. Each of these neurons is associated with a numerical value, known as bias.

Which is then added to the input sum before passing to the threshold function, called the activation function. An activated neuron is then used to determine if the particular neuron will be activated or not.

An activated neuron transmits data to the next layer of neurons over the channel.

And the data themselves are propagated through the network. This is called Forward Propagation.

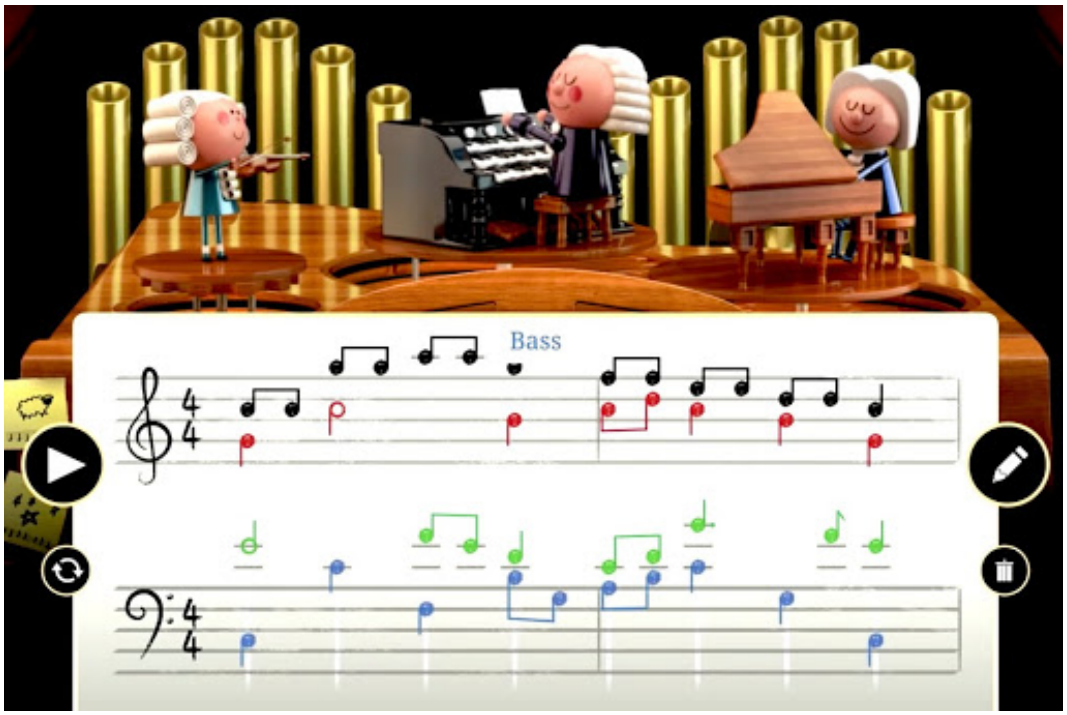
The neuron with the highest values will be used to determine the output

Then the predicted output is compared to the actual output to calculate the error. This information is transferred backward to modify the weight based on this. This is called backpropagation.

The process is iteratively done until the error is negligible or the prediction is accurate enough.

Some practical applications of the artificial neural network are music Composition, real-time translation and facial recognition.


Face recognition
Introducing face recognition for more features
Hi, Russell. We're always working to make Faceboo
adding more ways to use face recognition apart fro
tags. For example, face recognition technology ca
• Find photos you're in but haven't been tagged in
• Help protect you against strangers who use your
• Tell people with visual impairments who's in your
You control face recognition. This setting is on, bu
any time, which applies to features we may add lat
–
The Facebook Team


Bass


SALIDA
DE
EMERGENCIA
EXIT
OF
EMERGENCY

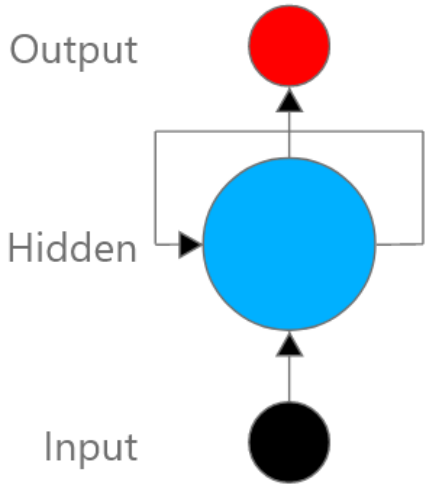
Recurrent Neural Network (RNN)

3.1 What's it about?

The RNN is a powerful technique evolved from the traditional neural network. It is good at dealing with sequential data. Therefore, it is commonly used in NLP, voice assistance space, stock prediction, text generation, etc.

The way that RNN handles sequential data is called sequential memory. For example, normally we spell the alphabet from A to Z easier than from Z back to A. RNN uses previous information to affect later one, by doing so it could recognize sequence patterns in the data.

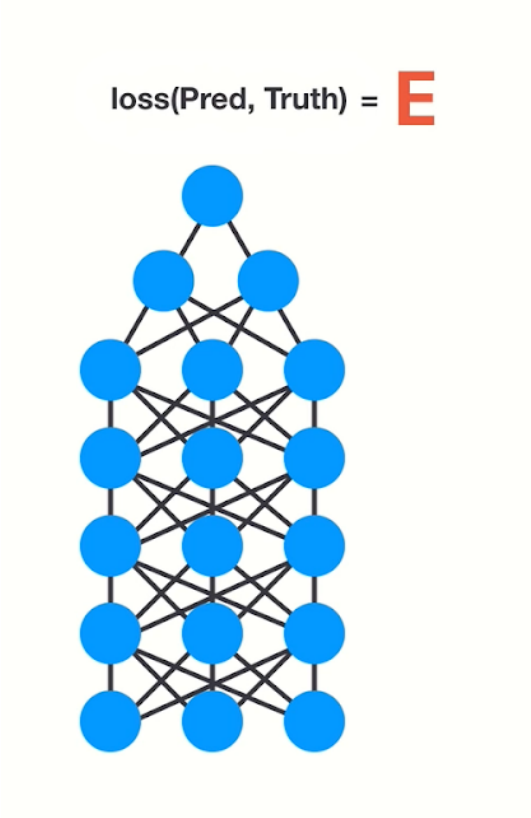
RNN has a loop in the neural network that can pass previous information forwards. This loop acts as a highway to allow information to flow from one step to the next.



3.2 Short-term memory and vanishing gradient problem

The vanishing gradient is a common difficulty in many RNNs in which the gradient is vanishingly small, causing the weight to be unchanged.

To demonstrate the problem, we need to look at three major steps of training the RNN: First it does a forward pass and makes a prediction(Pred), second it compares the prediction to the ground truth using a loss function.The loss function output is an error value (E) which is an estimate of how badly the net work is performing.



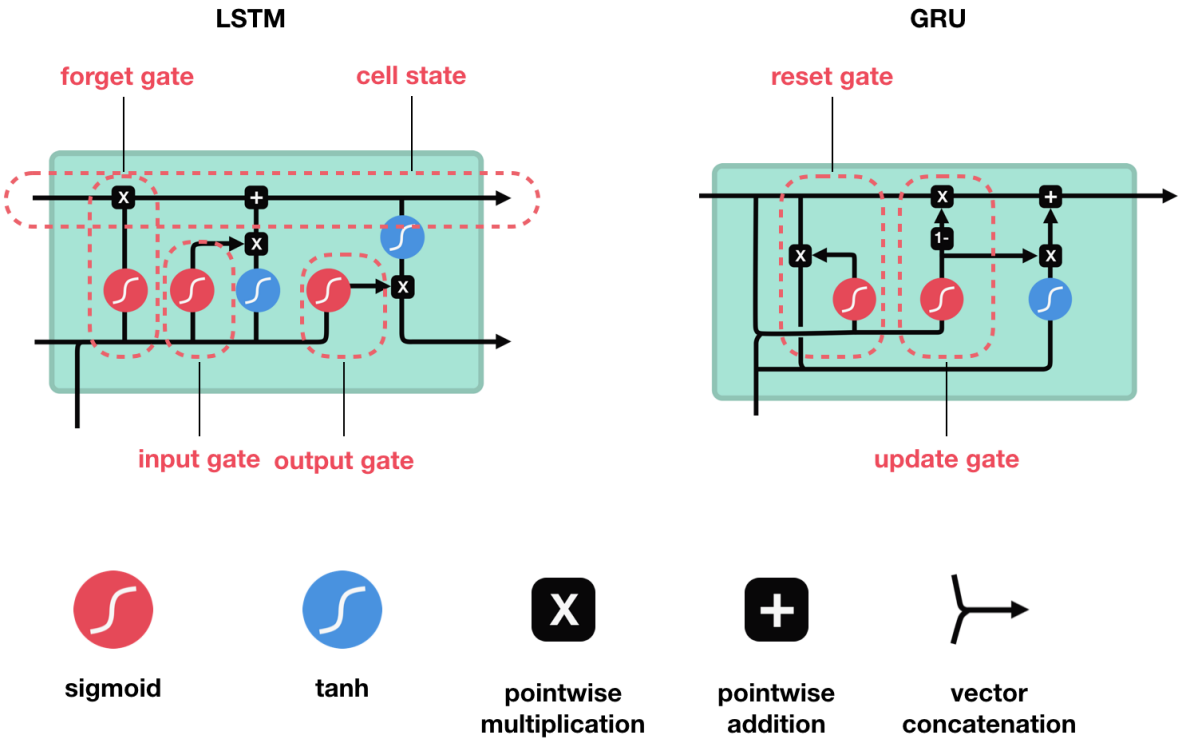
Lastly, it uses the error value to do back propagation which calculates the gradients for each node in the network. The gradient is the value used to adjust the network's internal weights allowing the network to learn. The bigger the gradient the bigger the adjustment.

When doing back propagation each node calculates its gradient with the respect to the effect of the gradients and the layer before , so the adjustment in the layers before is small, then the adjustment in the current layer will be even smaller.

This costs gradients to exponentially shrink as it back propagates down. The earlier layers failed to do anything as the internal weights are being adjusted due to extremely small gradients.

Long Short Term Memory (LSTM)

To deal with the Vanishing Gradient problem of the RNN during the process of Back Propagation, an evolved version of the artificial neuron network called Long Short Term Memory (LSTM) is created. It has successfully dealt with the problem using a mechanism named Gates.



The core concept of LSTM is its cell state and its various gates. The cell state acts like the memory of the network, it carries information throughout the sequence processing so that the information from earlier time steps could be delivered to the final ones, in consequence, reducing the effect of short-term memory.

The information is added or removed via gates. The gates will decide which information is relevant to keep or discard during training. Each gate contains a sigmoid function (value is between 0 and 1), so if the value is close to 0, it will be deleted and vice versa.

Weather Forecasting Implementation

This program uses the artificial recurrent network named Long Short Term Memory (LSTM) to predict temperature hourly using the last 30 data points.

First, we import all needed libraries for programming including sklearn, keras for core algorithm implementation; numpy for math and calculation; pandas for data pre/post-processing and finally matplotlib for visualization.

```
#Import the libraries
import math
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
```

5.1 Preprocess Data

Before the implementation of the model, we need to preprocess the raw data, beginning with creating the datetime object and set it to be the index column of the dataset.

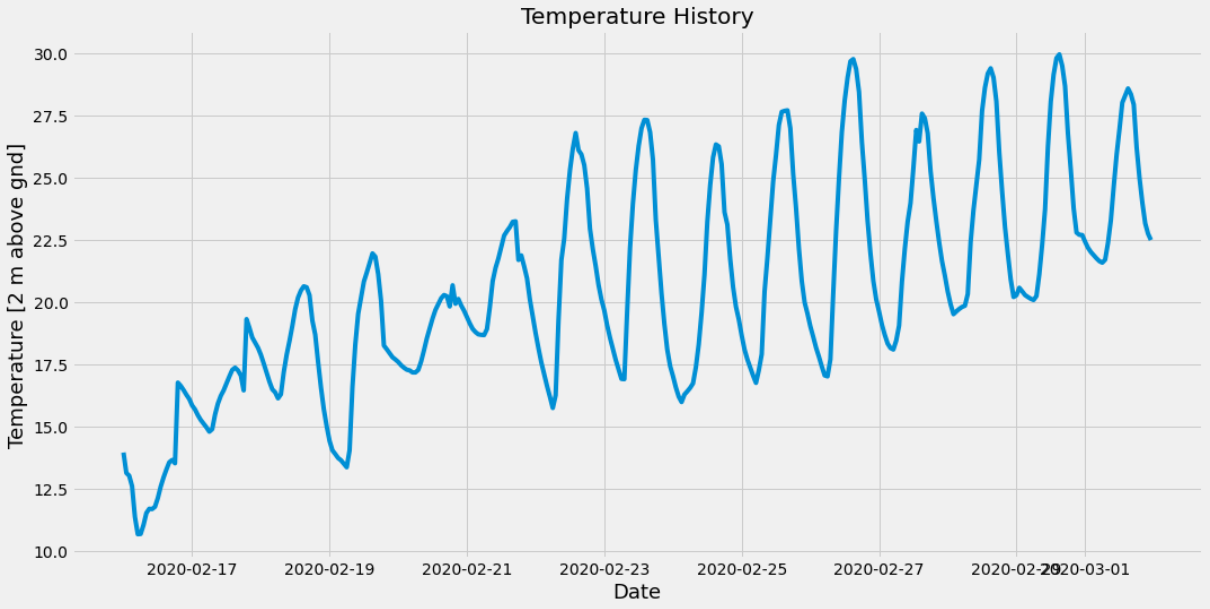
```
# Read the data frame and save it

df = pd.read_csv("hourly.csv", delimiter=';', skiprows=11)
df.head(5)
```

	Year	Month	Day	Hour	Minute	Temperature [2 m above gnd]
0	2020	2	16	0	0	13.94
1	2020	2	16	1	0	13.12
2	2020	2	16	2	0	13.01
3	2020	2	16	3	0	12.59
4	2020	2	16	4	0	11.35

```
# Create datetime columns, set to indexes
df['Date']=pd.to_datetime(df[['Year','Month','Day','Hour','Minute']])
df.set_index("Date", inplace=True)

# Get rid of these columns
df.drop(['Year','Month','Day','Hour','Minute'], axis=1, inplace=True)
```

Date	Temperature [2 m above gnd]
2020-02-16 00:00:00	13.94
2020-02-16 01:00:00	13.12
2020-02-16 02:00:00	13.01
2020-02-16 03:00:00	12.59
2020-02-16 04:00:00	11.35

Then, we visualize the "Temperature [2m above grid]" (our training data) using matplotlib. The temperature values are on the y-axis while the datetime is on the x-axis.



Overall, the temperature values are recorded hourly so the temperature is expected to be fluctuated during the day time (cold at night, warm at noon). The visualization sets up an expectation for our model to predict the data following an upward fluctuation.

Afterwards, we extract the temperature column to a new dataframe, using that for the LSTM implementation. The second preprocessing step is to scale the data into range 0 and 1 to be able to properly feed to the model.

```
#Create a new dataframe with only the 'Temp' column
data = df.filter(['Temperature  [2 m above gnd]'])
#Convert the dataframe to a numpy array
dataset = data.values
#Get the number of rows to train the model on
training_data_len = math.ceil( len(dataset) * .8 )
training_data_len

=> Output: 288
```

```
#Scale the data (between 0 and 1)
scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(dataset)

scaled_data
```

```
array([[1.70124481e-01],
       [1.27593361e-01],
       [1.21887967e-01],
       [1.00103734e-01],
       [3.57883817e-02],
       [0.00000000e+00],
       [5.18672199e-04],
       [1.91908714e-02],
       [4.40871369e-02],
       [5.29045643e-02],
       [5.23858921e-02],
       [5.70539419e-02]
```

Due to the fact that the training data has only 288 data points (80% of the dataset), we decided to choose 30 data points for each prediction step. The below code is to create the training data format to feed to the model as follows: 30 data points starting for the first value is appended to the x_train array, the 31st value is appended to the y_train array; then come 30 data starting from the second value to the x_train and the 32nd value to the y_train; so on.

```
#Create the training data set
#Create the scaled training data set
train_data = scaled_data[0:training_data_len , :]
#Split the data into x_train and y_train data sets
x_train = []
y_train = []

for i in range(30, len(train_data)):
  x_train.append(train_data[i-30:i, 0])
  y_train.append(train_data[i, 0])
  if i<= 31:
    print(x_train)
    print(y_train)
    print()
```

```
[array([0.17012448, 0.12759336, 0.12188797, 0.10010373, 0.03578838,
       0.        , 0.00051867, 0.01919087, 0.04408714, 0.05290456,
       0.05238589, 0.05705394, 0.07520747, 0.09906639, 0.11825726,
       0.13485477, 0.14989627, 0.15456432, 0.14782158, 0.31587137,
       0.30912863, 0.3003112 , 0.29045643, 0.281639  , 0.26815353,
       0.25985477, 0.24792531, 0.23807054, 0.23029046, 0.22251037])]
[0.2136929460580913]

[array([0.17012448, 0.12759336, 0.12188797, 0.10010373, 0.03578838,
       0.        , 0.00051867, 0.01919087, 0.04408714, 0.05290456,
       0.05238589, 0.05705394, 0.07520747, 0.09906639, 0.11825726,
       0.13485477, 0.14989627, 0.15456432, 0.14782158, 0.31587137,
       0.30912863, 0.3003112 , 0.29045643, 0.281639  , 0.26815353,
       0.25985477, 0.24792531, 0.23807054, 0.23029046, 0.22251037]), array([0.12759336, 0.12188797, 0.10010373, 0.03578838, 0.
       0.00051867, 0.01919087, 0.04408714, 0.05290456, 0.05238589,
       0.05705394, 0.07520747, 0.09906639, 0.11825726, 0.13485477,
       0.14989627, 0.15456432, 0.14782158, 0.31587137, 0.30912863,
       0.3003112 , 0.29045643, 0.281639  , 0.26815353, 0.25985477,
       0.24792531, 0.23807054, 0.23029046, 0.22251037, 0.21369295])]
[0.2136929460580913, 0.21887966804979264]
```

We finish the step by converting the array into numpy array and reshape the data, preparing to feed to the model.

```
#Convert the x_train and y_train to numpy arrays
x_train, y_train = np.array(x_train), np.array(y_train)
#Reshape the data
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
x_train.shape

=> Output: (258, 30, 1)
```

5.2 Build LSTM model

The architecture of the LSTM model includes 2 LSTM layers with 50 neurons each, followed by 2 Dense layers with 25 and 1 neurons respectively.

The chosen optimizer is 'adam' and the loss is calculated by mean squared error.

Then we train the model using our data via model.fit(). The batch size (total number of present training examples in a batch) and the epoch number (number of iteration when passing forwards) are set to 1.

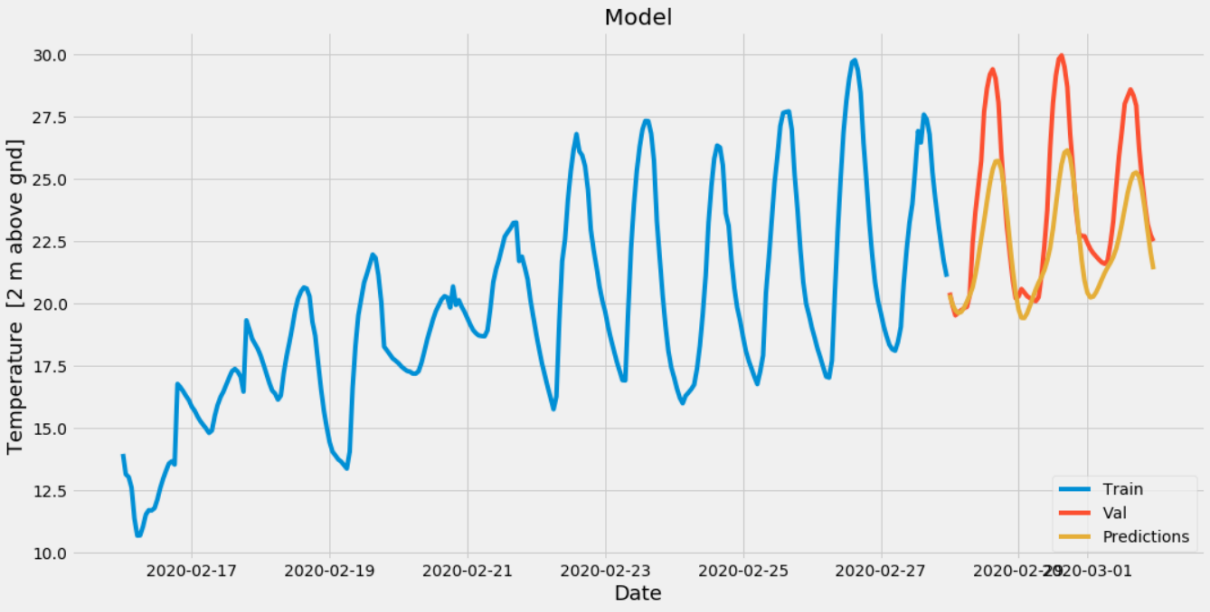
```
#Build the LSTM model
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape= (x_train.shape[1], 1)))
model.add(LSTM(50, return_sequences= False))
model.add(Dense(25))
model.add(Dense(1))
#Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

=> Output: WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras
    /optimizers.py:793: The name tf.train.Optimizer is deprecated. Please use tf
    .compat.v1.train.Optimizer instead.
```

5.3 Testing and Evaluation

After training our model using 288 data points, we use the 20% remaining values to test the model.

The testing data format is the same as the training one (30 in x_test-1 in y_test iteratively), then we use the Root Mean Squared Error to evaluate the model (2.297 is a decent score, presenting the efficiency of the model).

Finally, we visualize the prediction against the real value to demonstrate the result of the model using matplotlib.



As could be seen from the graph, the prediction values, although are not close to the real values at peaks, but they follow the same fluctuation pattern and the trough values are almost identical.