Abstract

Lets say you are going to an exposition filled with paints and you are trying to find the special paint

that has the more value. You will be looking at the shapes, colors, how ancient it is, how rare it is

and how many people are willing to get that rare paint. Trying to invest to invest into the stock market

is similar to look for that special paint. The question will be which company or shares that will make

you get more profit. Investing in stocks or shares has always been a controversial topic. Because if

there are people knowing exactly which company to invest, when to invest or which shares to buy now or

in the future, those people will be the richest in the world. Fortunately, nowadays there is a hope that

technology(AI) will help fullfil that goal in a way that will be described throughout this analysis.

Introduction:

AI can be found everywhere on apps for example when you are using Uber, google maps. Those apps use a

technology called self organizing map which is just an algorithm that organize things in a way it will

help you find the shortest route to go from point A to B. However, our interest will be in a different

type of machine learning algorithm called neural network but some insights must be given. Predicting

stock market price fall into the time series forecasting problem. In that problem the time is considered

as a dependent continous variable for some reasons. For example, the price of a stock can be updated at

any second and also the price of a stock or shares on a day may depend of the price of that stock the

day before. Because of the dependencies between the prices and time, a sequential model which will be

LSTM(Long short term memory) will be used with keras(Library that contains predefined models and neural

network). LSTM network is a type of recurrent neural network that remembers long term dependencies between the data.

The steps will be defined as follow:

-Defining the variables use as input

-Statings the assumptions or common beliefs about the model

- Data preprocessing using sklearn and numpy which are common libraries used for this task
- Data manipulation

```
# Numpy is used for data manipulation
import numpy as np

# Import math library to use sqrt functions
import math

# Matplot lib is used to visualize the data
import matplotlib.pyplot as plt

# Pandas to read to data
import pandas as pd

# sklearn for data preprocessing
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean_squared_error

# For data manipulation
from itertools import chain

#keras to create LSTM network and compile it later on
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.callbacks import Callback
from keras.models import Sequential
from keras.layers import LSTM, Dense, Activation

import time
import tensorflow as tf

import types
```

There are 2 datasets. 1 for training and the other to test how well the model perform to predict future prices. All datasets comes from **finance.yahoo.com**

```
from google.colab import files
uploaded = files.upload()
```

Choose Files Google_Stoc…_Train.csv

- **Google_Stock_Price_Train.csv**(application/vnd.ms-excel) - 63488 bytes, last modified: 9/7/2020 - 100%

```
df_data = pd.read_csv('Google_Stock_Price_Train.csv')
df_data.iloc[1:5,]
```

	Date	Open	High	Low	Close	Volume
1	1/4/2012	331.27	333.87	329.08	666.45	5,749,400
2	1/5/2012	329.83	330.75	326.89	657.21	6,590,300
3	1/6/2012	328.34	328.77	323.68	648.24	5,405,900
4	1/9/2012	322.04	322.29	309.46	620.76	11,688,800

```
# Test to see if a GPU is currently used for computations
device_name = tf.test.gpu_device_name()
if device_name != '/device:GPU:0':
    raise SystemError('GPU not found')
print('Found GPU at: {}'.format(device_name))
```

```
Found GPU at: /device:GPU:0
```

Let **X** represents the highest daily price from 05/24/2019 to 06/24/2019. X is also the input vector that will be feed into the neural network.

```
X = df_data.iloc[:,[2]].values
```

```
training_set = []

for i in range(1237):
    training_set.append(X[i])

training_set[1:5]
```

```
[array([333.87]), array([330.75]), array([328.77]), array([322.29])]
```

Cross-Validation

Given any dataset, it is important to decide which part of the dataset to train and which part will be used to test the model. The process of choosing how to split the dataset in training and testing part is called **cross validation.** They are several types of cross validation. However, the **K-fold valdation** will be used because it is most common and most known one. It consists in spliting the data into 5 parts. 80 % for trainining it is about 1006.4. So we will round to 1006. 20% percent will be used to test the model. It is about 252 entries. The dataset will be divided into 5 folds of 20 % of the total dataset each.

```
# Number of folds used
numb_folds = 5

# Create an array for 5 folds
folds      =  []

for i in range (1,6):
  curr_fold  = []

  for j in range(251):
    curr_fold.append(X[i*j])
  folds.append(curr_fold)
# Append the 3 remaining values from 1255 to 1258
#for k in range(8):
#  folds[4].append(X[1250+k])
```

```
#training_set = folds[1] +folds[2] +folds[3] +folds[4]
#testing_set  = folds[0]
```

```
testing_set  = []

for i in range(1238,1258):
    testing_set.append(X[i])
```

```
minmaxscaler = MinMaxScaler(feature_range=(0,1))
training_set_scaled    = minmaxscaler.fit_transform(training_set)
training_set_scaled[1:5]
```

```
array([[0.09834351],
       [0.09251685],
       [0.08881917],
       [0.07671765]])
```

```
testing_set_scaled    = minmaxscaler.fit_transform(testing_set)
```

```
# Create X_train and Y_train. X_train is the final input that will be feed into the neural ne
X_train = []
Y_train = []
timesteps = 60

for i in range(timesteps, len(training_set)):
    X_train.append(training_set_scaled[i-timesteps:i, 0])
    Y_train.append(training_set_scaled[i, 0])
X_train, Y_train = np.array(X_train), np.array(Y_train)
```

Show **X_train**

```
X_train[1:5,1:5]
```

```
array([[0.09251685, 0.08881917, 0.07671765, 0.06444805],
       [0.08881917, 0.07671765, 0.06444805, 0.06033951],
       [0.07671765, 0.06444805, 0.06033951, 0.063589  ],
       [0.06444805, 0.06033951, 0.063589  , 0.05806114]])
```

```
np.shape(X_train)
```

```
(1177, 60)
```

```
np.shape(Y_train)
```

```
(1177,)
```

```
# Reshaping
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
np.shape(X_train)
```

```
(1177, 60, 1)
```

```
model = Sequential()

model.add(LSTM(units = 60, return_sequences = True, input_shape = (X_train.shape[1], 1)))
model.add(Dropout(0.2))

# Adding a second LSTM layer and some Dropout regularisation
model.add(LSTM(units = 60, return_sequences = True))
model.add(Dropout(0.2))

# Adding a third LSTM layer and some Dropout regularisation
model.add(LSTM(units = 60, return_sequences = True))
model.add(Dropout(0.2))

# Adding a fourth LSTM layer and some Dropout regularisation
model.add(LSTM(units = 60))
model.add(Dropout(0.2))

# Adding the output layer
model.add(Dense(units = 1))

# Compiling the RNN
model.compile(optimizer = 'adam', loss = 'mean_squared_error')
```

```
model.fit(X_train, Y_train, epochs = 100, batch_size = 32)
```

```
Epoch 1/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0363
Epoch 2/100
37/37 [==============================] - 2s 56ms/step - loss: 0.0049
Epoch 3/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0047
Epoch 4/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0046
Epoch 5/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0054
Epoch 6/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0046
Epoch 7/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0037
Epoch 8/100
37/37 [==============================] - 2s 57ms/step - loss: 0.0044
Epoch 9/100
37/37 [==============================] - 2s 56ms/step - loss: 0.0039
Epoch 10/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0038
Epoch 11/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0038
Epoch 12/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0036
Epoch 13/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0036
Epoch 14/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0037
Epoch 15/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0036
Epoch 16/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0033
Epoch 17/100
37/37 [==============================] - 2s 56ms/step - loss: 0.0033
Epoch 18/100
37/37 [==============================] - 2s 56ms/step - loss: 0.0033
Epoch 19/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0039
Epoch 20/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0032
Epoch 21/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0029
Epoch 22/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0030
Epoch 23/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0030
Epoch 24/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0032
Epoch 25/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0031
Epoch 26/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0036
Epoch 27/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0033
Epoch 28/100
37/37 [==============================] - 2s 56ms/step - loss: 0.0027
Epoch 29/100
```

```
37/37 [==============================] - 2s 56ms/step - loss: 0.0025
Epoch 30/100
37/37 [==============================] - 2s 54ms/step - loss: 0.0028
Epoch 31/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0028
Epoch 32/100
37/37 [==============================] - 2s 54ms/step - loss: 0.0027
Epoch 33/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0027
Epoch 34/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0026
Epoch 35/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0026
Epoch 36/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0024
Epoch 37/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0026
Epoch 38/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0025
Epoch 39/100
37/37 [==============================] - 2s 56ms/step - loss: 0.0025
Epoch 40/100
37/37 [==============================] - 2s 54ms/step - loss: 0.0024
Epoch 41/100
37/37 [==============================] - 2s 54ms/step - loss: 0.0024
Epoch 42/100
37/37 [==============================] - 2s 56ms/step - loss: 0.0026
Epoch 43/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0022
Epoch 44/100
37/37 [==============================] - 2s 56ms/step - loss: 0.0021
Epoch 45/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0022
Epoch 46/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0022
Epoch 47/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0023
Epoch 48/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0021
Epoch 49/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0021
Epoch 50/100
37/37 [==============================] - 2s 56ms/step - loss: 0.0021
Epoch 51/100
37/37 [==============================] - 2s 56ms/step - loss: 0.0023
Epoch 52/100
37/37 [==============================] - 2s 56ms/step - loss: 0.0023
Epoch 53/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0023
Epoch 54/100
37/37 [==============================] - 2s 56ms/step - loss: 0.0023
Epoch 55/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0021
Epoch 56/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0021
Epoch 57/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0019
Epoch 58/100
```

```
37/37 [==============================] - 2s 56ms/step - loss: 0.0022
Epoch 59/100
37/37 [==============================] - 2s 56ms/step - loss: 0.0021
Epoch 60/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0018
Epoch 61/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0018
Epoch 62/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0018
Epoch 63/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0018
Epoch 64/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0018
Epoch 65/100
37/37 [==============================] - 2s 56ms/step - loss: 0.0018
Epoch 66/100
37/37 [==============================] - 2s 54ms/step - loss: 0.0017
Epoch 67/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0019
Epoch 68/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0018
Epoch 69/100
37/37 [==============================] - 2s 56ms/step - loss: 0.0017
Epoch 70/100
37/37 [==============================] - 2s 56ms/step - loss: 0.0017
Epoch 71/100
37/37 [==============================] - 2s 56ms/step - loss: 0.0016
Epoch 72/100
37/37 [==============================] - 2s 56ms/step - loss: 0.0018
Epoch 73/100
37/37 [==============================] - 2s 56ms/step - loss: 0.0017
Epoch 74/100
37/37 [==============================] - 2s 57ms/step - loss: 0.0016
Epoch 75/100
37/37 [==============================] - 2s 56ms/step - loss: 0.0017
Epoch 76/100
37/37 [==============================] - 2s 56ms/step - loss: 0.0016
Epoch 77/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0017
Epoch 78/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0018
Epoch 79/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0015
Epoch 80/100
37/37 [==============================] - 2s 56ms/step - loss: 0.0015
Epoch 81/100
37/37 [==============================] - 2s 56ms/step - loss: 0.0015
Epoch 82/100
37/37 [==============================] - 2s 56ms/step - loss: 0.0014
Epoch 83/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0016
Epoch 84/100
37/37 [==============================] - 2s 54ms/step - loss: 0.0016
Epoch 85/100
37/37 [==============================] - 2s 54ms/step - loss: 0.0015
Epoch 86/100
37/37 [==============================] - 2s 55ms/step - loss: 0.0016
Epoch 87/100
```

```
    37/37 [==============================] - 2s 55ms/step - loss: 0.0012
    Epoch 88/100
    37/37 [==============================] - 2s 56ms/step - loss: 0.0014
    Epoch 89/100
    37/37 [==============================] - 2s 55ms/step - loss: 0.0013
    Epoch 90/100
    37/37 [==============================] - 2s 55ms/step - loss: 0.0013
    Epoch 91/100
    37/37 [==============================] - 2s 55ms/step - loss: 0.0014
    Epoch 92/100
    37/37 [==============================] - 2s 55ms/step - loss: 0.0015
    Epoch 93/100
    37/37 [==============================] - 2s 55ms/step - loss: 0.0012
    Epoch 94/100
    37/37 [==============================] - 2s 56ms/step - loss: 0.0013
    Epoch 95/100
    37/37 [==============================] - 2s 55ms/step - loss: 0.0012
    Epoch 96/100
    37/37 [==============================] - 2s 55ms/step - loss: 0.0013
    Epoch 97/100
    37/37 [==============================] - 2s 56ms/step - loss: 0.0013
    Epoch 98/100
    37/37 [==============================] - 2s 55ms/step - loss: 0.0012
    Epoch 99/100
```

```
inputs = X[len(X) - len(testing_set) - timesteps:]
inputs = inputs.reshape(-1,1)
inputs = minmaxscaler.transform(inputs)
inputs[1:5]
```

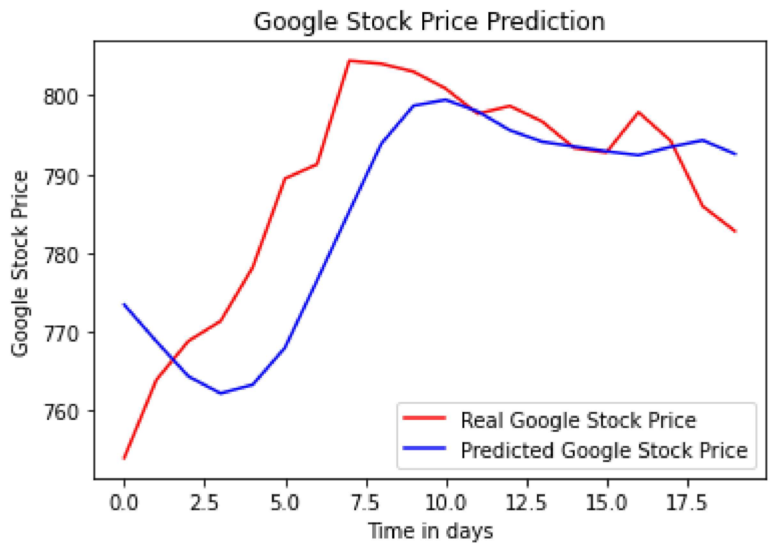
```
array([[0.38189758],
       [0.3233426 ],
       [0.24255657],
       [0.27153632]])
```

```
X_test = []
for i in range(timesteps, 80):
    X_test.append(inputs[i-timesteps:i, 0])
X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
```

```
predicted_stock_price = model.predict(X_test)
predicted_stock_price = minmaxscaler.inverse_transform(predicted_stock_price)
predicted_stock_price[1:5]
```

```
array([[768.77966],
       [764.34076],
       [762.1884 ],
       [763.295  ]], dtype=float32)
```

```
# Plot the results
plt.plot(testing_set, color = 'red', label = 'Real Google Stock Price')
plt.plot(predicted_stock_price, color = 'blue', label = 'Predicted Google Stock Price')
plt.title('Google Stock Price Prediction')
plt.xlabel('Time in days')
plt.ylabel('Google Stock Price')
plt.legend()
plt.show()
```



Computing RMSE

In Python, the following procedure can be used:

- Compute Mean Square Error using a built in function from sklearn. metrics. mean_squared_error
- Compute the sqrt of MSE

```
MSE = mean_squared_error(predicted_stock_price, testing_set)
RMSE = math.sqrt(MSE)
RMSE
```

```
10.260683392275405
```

Conclusion

We notice over time that the two curves have similar shape overtime. Also, the low RMSE indicates that there is a small difference between the predicted and actual price. That difference might be explained to the choice of parameters before training the model or simply due to underfitting.