

# LSTM Recurrent Neural Networks to Predict Google Stock Prices

This project involves the use of Recurrent Neural Networks along with Long Short-Term Memory (LSTM) to predict the stock prices of Google.

## About RNN

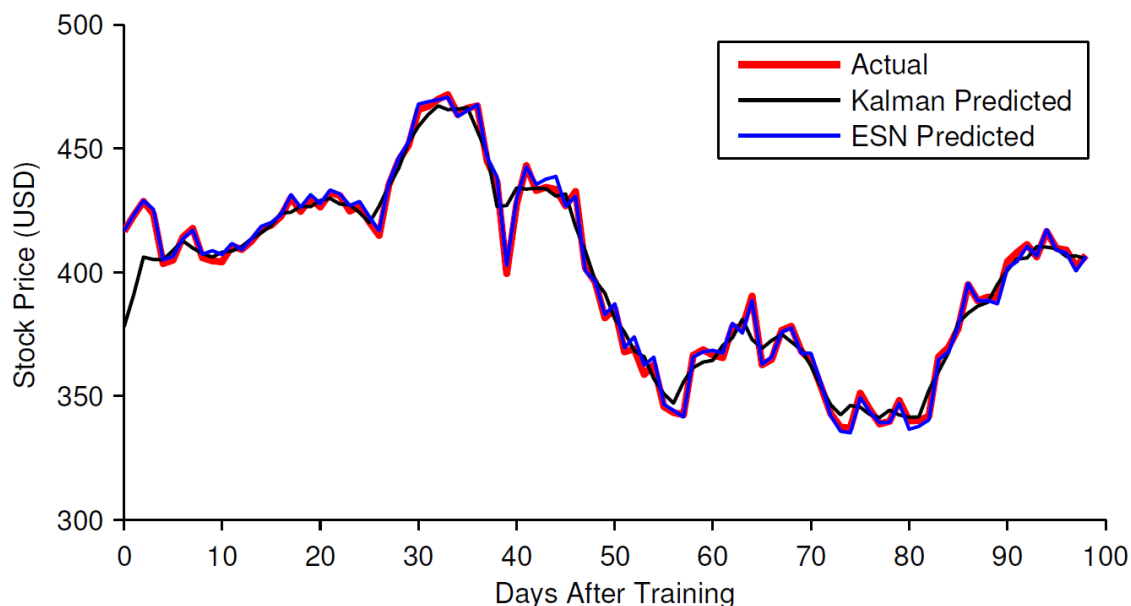
Recurrent Neural Network (RNN) was designed, like any other Neural Network, to function like a specific part of the human brain. When looking at our brain's cerebrum, we can divide it into the Temporal, Parietal, Occipital and Frontal lobe. Out of these, the Frontal lobe is the part which deals with short-term memory and remembers what happened in the immediate present and use it for decision making in the near future. It is this Frontal lobe that the RNN tries to replicate.

## About LSTM

The Long Short-Term Memory (LSTM) is a variation of the RNN which solves the Vanishing Gradient Problem, which leads to the decrease in gradient for each level in a regular RNN because of the Recurring Weight being close to 0. This leads to a difficulty in the manipulation of weights for a layer farther away from the present layer and thus makes predictions inaccurate. We would not go into the details of this phenomenon.

## About the Dataset

We are using Google's Stock price from 5 years till now from a financial website (Yahoo! Finance). The idea of this project was based on a project by students from Stanford (Financial Market Time Series Prediction with Recurrent Neural Networks - Bernal, Fok, Pidaparthi). The team used an Echo State Network instead of an LSTM. We will use their findings as a comparison to how our LSTM performs. The team trained their model from late 2004 to early 2009 with data from Yahoo! Finance. They created a visualization comparing their predictions with the actual data.



We will also train our LSTM on 5 years of data. We can see that their predictions are quite close to the actual Stock Price. We can try to get the same accuracy from our model as well.

We assume that the present day is January 01, 2017. We will then get the Google Stock price for the previous 5 years. Once we train our LSTM, we will try to predict the stock price for the month of January 2017.

## Data Preprocessing

We have the training set of 5 years of Google Stock price. The test set contains the stock price for January 2017. We will first import it. As we can see in the dataframe, the dates range from 2012 to 2016.

The screenshot shows the Spyder Python IDE with the following code in the editor:

```
1# Recurrent Neural Network
2
3# Part 1 - Data Preprocessing
4
5# Importing the libraries
6import numpy as np
7import matplotlib.pyplot as plt
8import pandas as pd
9
10# Importing the training set
11# Just predicting the "Open Stock Price" for Google. So extracting 1 column.
12training_set = pd.read_csv('Google_Stock_Price_Train.csv')
13training_set = training_set.iloc[:,1:2].values
14
15# Feature Scaling
16from sklearn.preprocessing import MinMaxScaler
17sc = MinMaxScaler()
18training_set = sc.fit_transform(training_set)
```

The Variable explorer shows the `training_set` DataFrame with 1258 rows and 6 columns: Date, Open, High, Low, Close, and Volume. The IPython console shows the execution of the code, including the import of libraries and the creation of the `training_set` DataFrame.

We can see that there are 3 types of stocks - Open, High, Low and Close. There is also a Volume column which contains the Volume of stocks for Google. We will focus on the Open stock price and will predict this price for January 2017.

The input for RNN will NOT be Date and Open Stock Price, it will just be the Open Stock Price for different time frames. We will get this by using just the Open Stock Price from our DataFrame using `iloc`.

The screenshot shows the Spyder Python IDE with the following code in the editor:

```
7import matplotlib.pyplot as plt
8import pandas as pd
9
10# Importing the training set
11# Just predicting the "Open Stock Price" for Google. So extracting 1 column.
12training_set = pd.read_csv('Google_Stock_Price_Train.csv')
13# To convert the Vector form of a single column into a Matrix form, we will use 1:2 as the column index.
14# The 2nd column will be ignored and we will get our Open Stock Price Column in a Matrix form.
15# Output will be a 2d Numpy array, exactly what we want.
16training_set = training_set.iloc[:,1:2].values
17
18# Feature Scaling
19from sklearn.preprocessing import MinMaxScaler
20sc = MinMaxScaler()
21training_set = sc.fit_transform(training_set)
22
23# Getting the inputs and the outputs
24X_train = training_set[0:1257]
25y_train = training_set[1:1258]
26
27# Reshaping
28X_train = np.reshape(X_train, (1257, 1))
29
30# Part 2 - Building the RNN
31
32# Importing the Keras libraries and
33from keras.models import Sequential
34from keras.layers import Dense
35from keras.layers import LSTM
36
37# Initialising the RNN
38regressor = Sequential()
39
40# Adding the input layer and the LSTM
41regressor.add(LSTM(units = 4, activation = 'tanh', input_shape = (1, 1)))
42
43# Adding the output layer
44regressor.add(Dense(units = 1))
45
46# Compiling the RNN
47regressor.compile(optimizer = 'adam')
48
49
```

The Variable explorer shows the `training_set` NumPy array with shape (1258, 1) and dtype float64. The IPython console shows the execution of the code, including the import of libraries and the creation of the `training_set` NumPy array.



The use of LSTM involves the prediction along the time of a particular value. The input that we have is currently 2-dimensional - we have 1257 rows and 1 column. We need to add another dimension to the input in account of time. This process is called reshaping. This format of input is required by Keras and the arguments have to be in the order of batch\_size(number of rows), timesteps(the number of time intervals or days between any 2 rows, in this case it will be 1) and input\_dim (number of columns). These 3 arguments are encapsulated together and come after the original data as the argument of Numpy's reshape function.

The screenshot shows the Spyder Python IDE interface. The main editor displays a Python script for loading and preprocessing stock price data. The code includes imports for matplotlib, pandas, and sklearn, followed by steps to load a CSV file, filter for training data, and reshape it into a 3D array for LSTM input. The file explorer on the right shows the project structure, and the variable explorer at the bottom shows the current state of variables in the workspace.

```
7 import matplotlib.pyplot as plt
8 import pandas as pd
9
10 # Importing the training set
11 # Just predicting the "Open Stock Price" for Google. So extracting 1 column.
12 training_set = pd.read_csv('Google_Stock_Price_Train.csv')
13 # Getting just the Open Stock Price for input of our RNN.
14 # To convert the Vector form of a single column into a Matrix form, we will use 1:2 as the column index.
15 # The 2nd column will be ignored and we will get our Open Stock Price Column in a Matrix form.
16 # Output will be a 2d numpy array, exactly what we want.
17 training_set = training_set.iloc[:,1:2].values
18
19 # Feature Scaling
20 # Will use Normalisation as the Scaling function.
21 # Default range for MinMaxScaler is 0 to 1, which is what we want. So no arguments in it.
22 # Will fit the training set to it and get it scaled and replace the original set.
23 from sklearn.preprocessing import MinMaxScaler
24 sc = MinMaxScaler()
25 training_set = sc.fit_transform(training_set)
26
27 # Getting the inputs and the outputs
28 # Restricting the input and output based on how LSTM functions.
29 X_train = training_set[0:1257]
30 y_train = training_set[1:1258]
31
32 # Reshaping - Adding time interval as a dimension for input.
33 X_train = np.reshape(X_train, (1257, 1, 1))
34
35 # Part 2 - Building the RNN
```

**File explorer:**

Name	Size	Type	Date Modified
ScreenShots		File Folder	8/27/2017 9:02 PM
.DS_Store	6 KB	DS_Store File	4/6/2017 9:25 PM
About the Project.txt	922 bytes	txt File	8/27/2017 8:51 PM
Google_Stock_Price_Test.csv	1 KB	csv File	4/3/2017 10:10 AM
Google_Stock_Price_Train.csv	62 KB	csv File	3/29/2017 3:42 PM
mn_evaluation.py	2 KB	py File	4/6/2017 9:29 PM
mn_homework_solution.py	2 KB	py File	4/6/2017 9:34 PM
mn.py	1 KB	py File	4/6/2017 9:29 PM

**Variable explorer:**

Name	Type	Size	Value
x_train	Float64	(1257, 1, 1)	array([[[ 0.08581368]],
training_set	Float64	(1258, 1)	array([[ 0.08581368],
y_train	float64	(1257, 1)	array([[ 0.09701243],

**Python console:**

```
1: import numpy as np
```

## Building the RNN

We will first import 3 classes. The Sequential class that will initialise our RNN. The Dense class will create the output layer of our RNN. And finally, the LSTM class which will make our RNN have "Long Memory".

The method to create the RNN will be the same as the one used to create ANN and CNN. We will then use other methods of the Dense and LSTM classes to add the layers and compile it, and eventually fit it.

We are predicting a continuous variable and are hence using a regression model instead of a classification model and call our object 'regressor'. To this regressor object, we will add the LSTM layer, which in itself will take the input layer as input. The arguments we add to the LSTM layer are: units - number of memory units, activation function - can be tanh or sigmoid. Other arguments will be default. But there will be an additional argument - the input shape argument to specify the format of our input layer. This argument would be none and 1 - none to specify that model can expect any time step and 1 because we have just 1 column of input. The optimal number of memory units that we can use is 4, the activation function is sigmoid, and the input\_shape would be (None, 1).

The next layer that we will add is the Output layer. We will use the Dense class, with the argument being units, everything else being default. The units are the number of neurons that should be present in the output layer, which is dependent on the dimensions of the output. So, our units argument for Dense class will have value 1.

To compile all the layers into a single system, we will use the compile function along with its arguments. Optimizer can be RMSprop or Adam. Both the optimizers gave similar results but RMS was memory heavy, so we went forward with Adam. But usually, RMSprop is recommended in Keras documentation. The Loss argument decides the manipulation of weights, so for this we should be using Mean Squared Error for continuous variable. For test set, we might use Root Mean Square Error in its place. Other arguments will be default.

```
File Edit Search Source Run Debug Consoles Projects Tools View Help
D:\Deep Learning A to Z\Krill Eremanko\Deep Learning A-Z\Volume 1 - Supervised Deep Learning\Part 3 - Recurrent Neural Networks (RNN)\Section 12 - Building a RNN\Recurrent_Neural_Networks
Editor - D:\Deep Learning A to Z\Krill Eremanko\Deep Learning A-Z\Volume 1 - Supervised Deep Learning\Part 3 - Recurrent Neural Networks (RNN)\Section 12 - Building a RNN\Recurrent_Ne...
File explorer
rmn.py* rmn_evaluation.py rmn_homework_solution.py
25 training_set = sc.fit_transform(training_set)
26
27 # Getting the inputs and the outputs
28 # Restricting the input and output based on how LSTM functions.
29 X_train = training_set[0:1257]
30 y_train = training_set[1:1258]
31
32 # Reshaping - Adding time interval as a dimension for input.
33 X_train = np.reshape(X_train, (1257, 1, 1))
34
35 # Part 2 - Building the RNN
36
37 # Importing the Keras libraries and packages
38 from keras.models import Sequential
39 from keras.layers import Dense
40 from keras.layers import LSTM
41
42 # Initialising the RNN
43 # Creating an object of Sequential class to create the RNN.
44 regressor = Sequential()
45
46 # Adding the input layer and the LSTM layer
47 # 4 memory units, sigmoid activation function and (None time interval with 1 attribute as input)
48 regressor.add(LSTM(units = 4, activation = 'sigmoid', input_shape = (None, 1)))
49
50 # Adding the output layer
51 # 1 neuron in the output layer for 1 dimensional output
52 regressor.add(Dense(units = 1))
53
54 # Compiling the RNN
55 # Compiling all the layers together.
56 # Loss helps in manipulation of weights in NN.
57 regressor.compile(optimizer = 'adam', loss = 'mean_squared_error')
58
```

Name	Type	Size	Date Modified
ScreenShots	File Folder		8/27/2017 9:02 PM
DS_Store	6 KB DS_Store File		4/6/2017 9:25 PM
About the Project.txt	922 bytes txt File		8/27/2017 8:51 PM
Google_Stock_Price_Test.csv	1 KB csv File		4/3/2017 10:10 AM
Google_Stock_Price_Train.csv	62 KB csv File		3/29/2017 3:42 PM
rmn_evaluation.py	2 KB py File		4/6/2017 9:29 PM
rmn_homework_solution.py	2 KB py File		4/6/2017 9:34 PM
rmn.py	1 KB py File		4/6/2017 9:29 PM

Name	Type	Size	Value
X_train	float64 (1257, 1, 1)	array([[ 0.08581368]],	
training_set	float64 (1258, 1)	array([[ 0.08581368],	
y_train	float64 (1257, 1)	array([[ 0.09701243],	
		[ 0.09433366]],	

```
File "c:\python-input-6-e029ff700625", line 1, in <module>
from keras.models import Sequential
ModuleNotFoundError: No module named 'keras'
```

Now we will fit this regressor to the training dataset. We will use the fit method for this. The important arguments include the input, output, batch\_size and epochs. We will keep the default batch size of 32 but will change epochs to 200 for better convergence.

The screenshot shows the Spyder Python IDE with the following components:

- Editor:** Contains Python code for fitting an RNN regressor. The code includes imports for Keras, reshaping of training data, and the `regressor.fit()` method call with `batch_size = 32` and `epochs = 200`.
- File explorer:** Shows the project structure with files like `mn.py`, `mn_evaluation.py`, and `mn_homework_solution.py`.
- Variable explorer:** Displays the variables `X_train`, `training_set`, and `y_train` with their respective shapes and values.
- Python console:** Shows the execution of the `regressor.compile()` and `regressor.fit()` methods, along with the training progress output showing loss decreasing over epochs.

With the passage of fitting, we see that the loss keeps on decreasing. But we will get accurate results only if we have the same loss in the test set.

The screenshot shows the Spyder Python IDE with the following components:

- Editor:** Contains Python code for evaluating the trained regressor. The code includes loading the test set, making predictions, and calculating the loss.
- Variable explorer:** Displays the variables `X_train`, `training_set`, and `y_train` with their respective shapes and values.
- Python console:** Shows the execution of the `regressor.compile()` and `regressor.fit()` methods, along with the training progress output showing loss decreasing over epochs. The final output shows the loss for the test set.



## Making Predictions and Visualising the Result

The methods of getting the Test Dataset is same as that Training Dataset. We will just rename it to `real_stock_price` so that we can distinguish between prediction and actual values.

Next, we use our model to make predictions on the test dataset. But we should keep in mind that every prediction is for the next day and not the present. The input will be the `real_stock_price`. The model that we have made is on scaled values. When used as it is, it will give incorrect predictions. So we will convert the input using the same “sc” object used for scaling the training data. We will also have to reshape the data according to the format expected by the predict method in a 3d format.

```
49
50# Adding the output Layer
51# 1 neuron in the output layer for 1 dimensional output
52 regressor.add(Dense(units = 1))
53
54# Compiling the RNN
55# Compiling all the layers together.
56# Loss helps in manipulation of weights in NN.
57 regressor.compile(optimizer = 'adam', loss = 'mean_squared_error')
58
59# Fitting the RNN to the Training set
60# Number of epochs increased for better convergence.
61 regressor.fit(X_train, y_train, batch_size = 32, epochs = 200)
62
63# Part 3 - Making the predictions and visualising the results
64
65# Getting the real stock price of 2017
66 test_set = pd.read_csv('Google_Stock_Price_Test.csv')
67 real_stock_price = test_set.iloc[:,1:2].values
68
69# Getting the predicted stock price of 2017
70 inputs = real_stock_price
71 inputs = sc.transform(inputs)
72 inputs = np.reshape(inputs, (20, 1, 1))
73 predicted_stock_price = regressor.predict(inputs)
74 predicted_stock_price = sc.inverse_transform(predicted_stock_price)
75
76# Visualising the results
77 plt.plot(real_stock_price, color = 'red', label = 'Real Google Stock Price')
78 plt.plot(predicted_stock_price, color = 'blue', label = 'Predicted Google Stock Price')
79 plt.title('Google Stock Price Prediction')
80 plt.xlabel('Time')
81 plt.ylabel('Google Stock Price')
82 plt.legend()
83 plt.show()
```

The variable explorer shows the following variables:

Name	Type	Size	Value
X_train	float64	(1257, 1, 1)	array([[[[ 0.08581368]]],
inputs	float64	(20, 1, 1)	array([[[[ 0.92955205]]],
real_stock_price	float64	(20, 1)	array([[ 778.81],
test_set	DataFrame	(20, 6)	Column names: Date, Open, High, Low, Close, Volume

The IPython console shows the following output:

```
1257/1257 [=====] - 0s - loss: 2.4998e-04
Epoch 196/200
1257/1257 [=====] - 0s - loss: 2.4741e-04
Epoch 197/200
1257/1257 [=====] - 0s - loss: 2.4969e-04
Epoch 198/200
1257/1257 [=====] - 0s - loss: 2.4788e-04
Epoch 199/200
1257/1257 [=====] - 0s - loss: 2.4898e-04
Epoch 200/200
1257/1257 [=====] - 0s - loss: 2.5114e-04
Out[12]: <keras.callbacks.History at 0x2385dd4aef0>

In [13]: test_set = pd.read_csv('Google_Stock_Price_Test.csv')
...: real_stock_price = test_set.iloc[:,1:2].values

In [14]: inputs = real_stock_price
...: inputs = sc.transform(inputs)
...: inputs = np.reshape(inputs, (20, 1, 1))
```

We will now use the regressor model to make predictions on the input and store it in the `predicted_stock_price`. The argument would obviously be the input. We now have the predicted stock price for the January 2017.

But this output will be scaled. We will have to use the inverse transform method of the same “sc” object we had used to scale the data to get the proper predicted values. This is the final prediction.

The Spyder Python IDE shows the following code in the editor:

```
35# Part 2 - building the RNN
36
37# Importing the Keras libraries and packages
38 from keras.models import Sequential
39 from keras.layers import Dense
40 from keras.layers import LSTM
41
42# Initialising the RNN
43# Creating an object of Sequential class to create the RNN
44 regressor = Sequential()
45
46# Adding the input layer and the LSTM layer
47# 4 memory units, sigmoid activation function
48 regressor.add(LSTM(units = 4, activation = 'sigmoid'))
49
50# Adding the output Layer
51# 1 neuron in the output layer for 1 dimensional output
52 regressor.add(Dense(units = 1))
53
54# Compiling the RNN
55# Compiling all the layers together.
56# Loss helps in manipulation of weights in NN.
57 regressor.compile(optimizer = 'adam', loss = 'mean_squared_error')
58
59# Fitting the RNN to the Training set
60# Number of epochs increased for better convergence.
61 regressor.fit(X_train, y_train, batch_size = 32, epochs = 200)
62
63# Part 3 - Making the predictions and visualising the results
64
65# Getting the real stock price of 2017
66 test_set = pd.read_csv('Google_Stock_Price_Test.csv')
67 real_stock_price = test_set.iloc[:,1:2].values
68
69# Getting the predicted stock price of 2017
70 inputs = real_stock_price
71 inputs = sc.transform(inputs)
72 inputs = np.reshape(inputs, (20, 1, 1))
73 predicted_stock_price = regressor.predict(inputs)
74 predicted_stock_price = sc.inverse_transform(predicted_stock_price)
75
```

The variable explorer shows the following variables:

Name	Type	Size	Value
X_train	float64	(1257, 1, 1)	array([[[[ 0.08581368]]],
inputs	float64	(20, 1, 1)	array([[[[ 0.92955205]]],
predicted_stock_price	float32	(20, 1)	array([[ 777.74841309],
real_stock_price	float64	(20, 1)	array([[ 778.81],

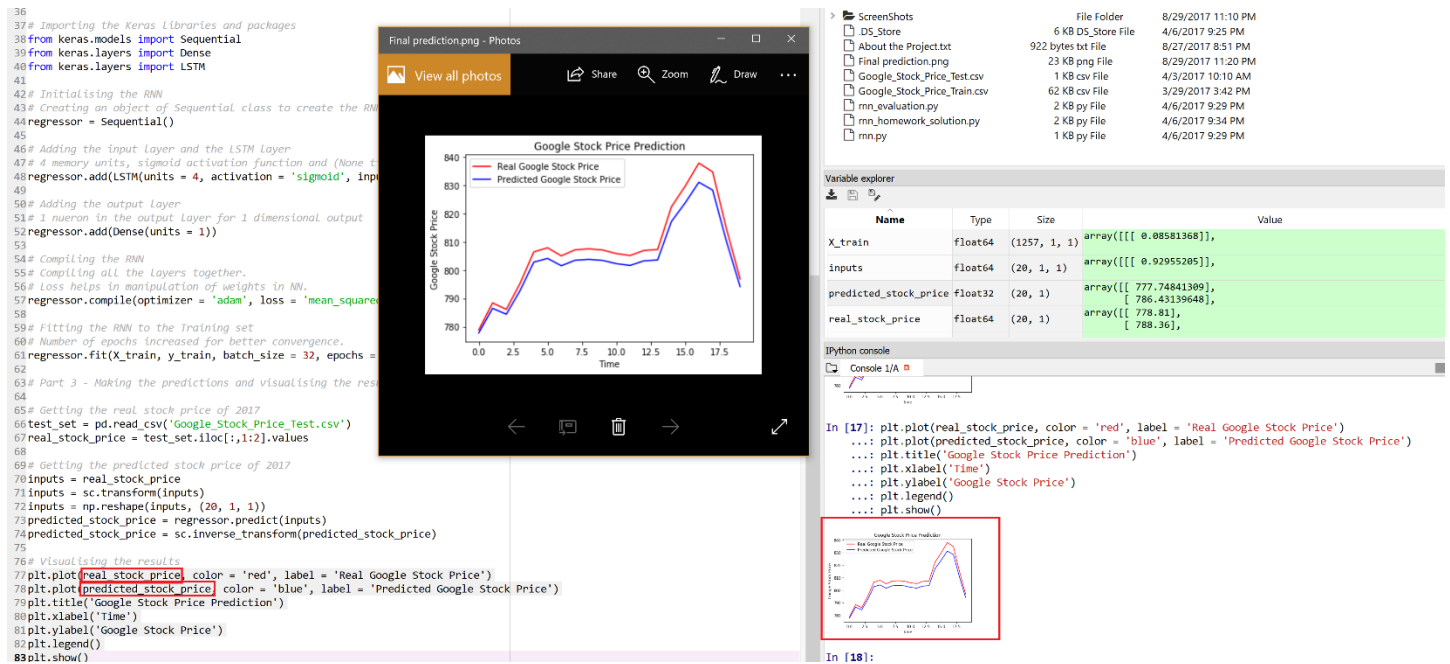
The IPython console shows the following output:

```
Epoch 197/200
1257/1257 [=====] - 0s - loss: 2.4969e-04
Epoch 198/200
1257/1257 [=====] - 0s - loss: 2.4788e-04
Epoch 199/200
1257/1257 [=====] - 0s - loss: 2.4898e-04
Epoch 200/200
1257/1257 [=====] - 0s - loss: 2.5114e-04
Out[12]: <keras.callbacks.History at 0x2385dd4aef0>

In [13]: test_set = pd.read_csv('Google_Stock_Price_Test.csv')
...: real_stock_price = test_set.iloc[:,1:2].values
```

A plot titled "Google Stock Price Prediction" is shown, displaying the real stock price (red line) and the predicted stock price (blue line) over time. The predicted stock price is shown as a NumPy array with values: [ 777.748, 786.431, 784.360, 792.692, 802.775, 804.094, 801.509, 803.443, 803.751, 803.389, 802.241, 801.618].

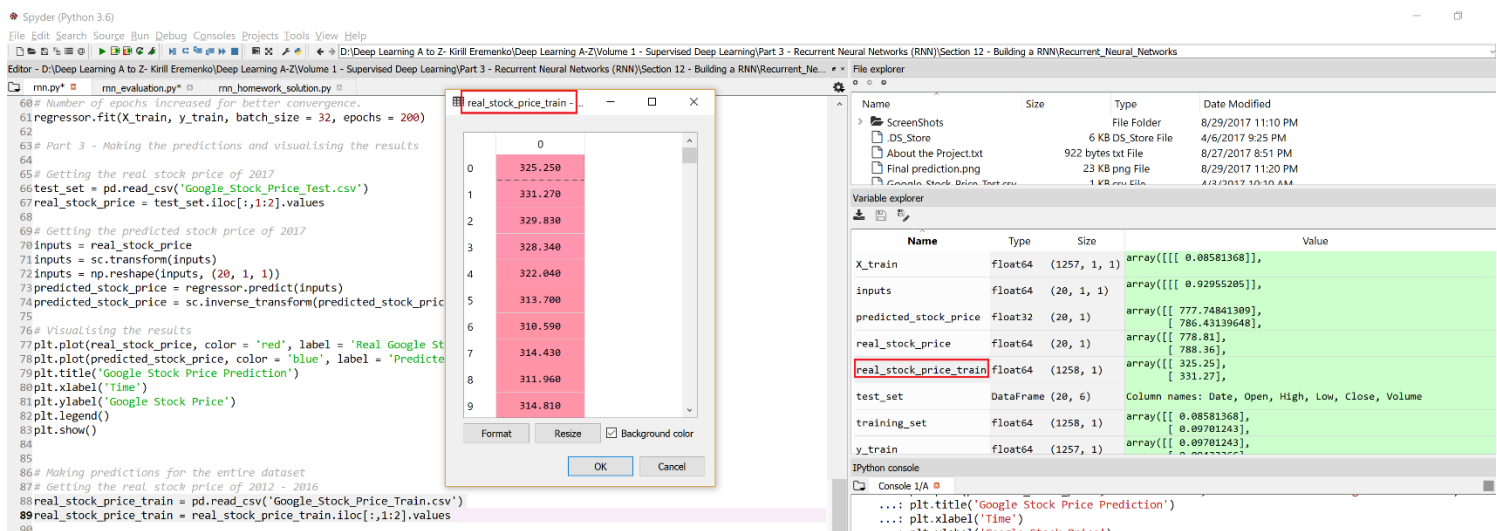
Now we visualize our predictions with the actual stock prices of Google. For this we use the pyplot module. We have some arguments with pyplot, like the use of color. For real stock prices, we will use Red. We will also include a label mentioning the real stock price. We will keep blue the color for Predicted Stock price and change the label as well. We will also add the axis labels and title and display it.



What's important to note is that this is a 1 time-step prediction i.e. input is of time  $t$  and prediction is of time  $t+1$ . We should note that we were able to make these predictions for 20 days only because we had the stock price for 20 days. This would not have been possible if we had the stock price for just 1 day. It would be wonderful to make predictions for a long future, but we would hardly get such amazing predictions. In finance, there is the Browning Motion, which makes future values of stock prices independent of the past, so it would be impossible to make long term predictions for stock price.

## Further Analysis

We will take this a little further and make predictions for the stock price from 2012 to 2016. The method of getting the input, scaling and reshaping has not changed.





We already have the regressor and Xtrain from previous work. We just have to make predictions using it. And then we can unscale the predictions to regain the actual values of the stocks.

Spyder (Python 3.6)

```

File Edit Search Source Run Debug Consoles Projects Tools View Help
D:\Deep Learning A to Z- Kirill Ermenko\Deep Learning A-Z\Volume 1 - Supervised Deep Learning\Part 3 - Recurrent Neural Networks (RNN)\Section 12 - Building a RNN\Recurrent_Neural_Networks
Editor - D:\Deep Learning A to Z- Kirill Ermenko\Deep Learning A-Z\Volume 1 - Supervised Deep Learning\Part 3 - Recurrent Neural Networks (RNN)\Section 12 - Building a RNN\Recurrent_Neural_Networks
rm.py* rm_evaluation.py* rm_homework_solution.py*
60 # Number of epochs increased for better convergence.
61 regressor.fit(X_train, y_train, batch_size = 32, epochs = 200)
62
63 # Part 3 - Making the predictions and visualising the results
64
65 # Getting the real stock price of 2017
66 test_set = pd.read_csv('Google_Stock_Price_Test.csv')
67 real_stock_price = test_set.iloc[:,1:2].values
68
69 # Getting the predicted stock price of 2017
70 inputs = real_stock_price
71 inputs = sc.transform(inputs)
72 inputs = np.reshape(inputs, (20, 1, 1))
73 predicted_stock_price = regressor.predict(inputs)
74 predicted_stock_price = sc.inverse_transform(predicted_stock_price)
75
76 # Visualising the results
77 plt.plot(real_stock_price, color = 'red', label = 'Real Google Stock Price')
78 plt.plot(predicted_stock_price, color = 'blue', label = 'Predicted Google Stock Price')
79 plt.title('Google Stock Price Prediction')
80 plt.xlabel('Time')
81 plt.ylabel('Google Stock Price')
82 plt.legend()
83 plt.show()
84
85
86 # Making predictions for the entire dataset
87 # Getting the real stock price of 2012 - 2016
88 real_stock_price_train = pd.read_csv('Google_Stock_Price_Train.csv')
89 real_stock_price_train = real_stock_price_train.iloc[:,1:2].values
90
91 # Getting the predicted stock price of 2012 - 2016
92 predicted_stock_price_train = regressor.predict(X_train)
93 predicted_stock_price_train = sc.inverse_transform(predicted_stock_price_train)

```

predicted\_stock\_price\_train

	0
0	326.361
1	332.236
2	330.830
3	329.375
4	323.233
5	315.121
6	312.102
7	315.830
8	313.431
9	316.200
10	313.606
11	320.566

Format Resize Background color OK Cancel

Variable explorer

Name	Type	Size	Value
X_train	float64	(1257, 1, 1)	array([[ 0.08581368]],
inputs	float64	(20, 1, 1)	array([[ 0.92955205]],
predicted_stock_price	float32	(20, 1)	array([[ 777.74841309],
predicted_stock_price_train	float32	(1257, 1)	array([[ 326.36148071],
real_stock_price	float64	(20, 1)	array([[ 778.81],
real_stock_price_train	float64	(1258, 1)	array([[ 325.25],
test_set	DataFrame	(20, 6)	Column names: Date, Open, High, Low, Close, Volume
training_set	float64	(1258, 1)	array([[ 0.08581368],

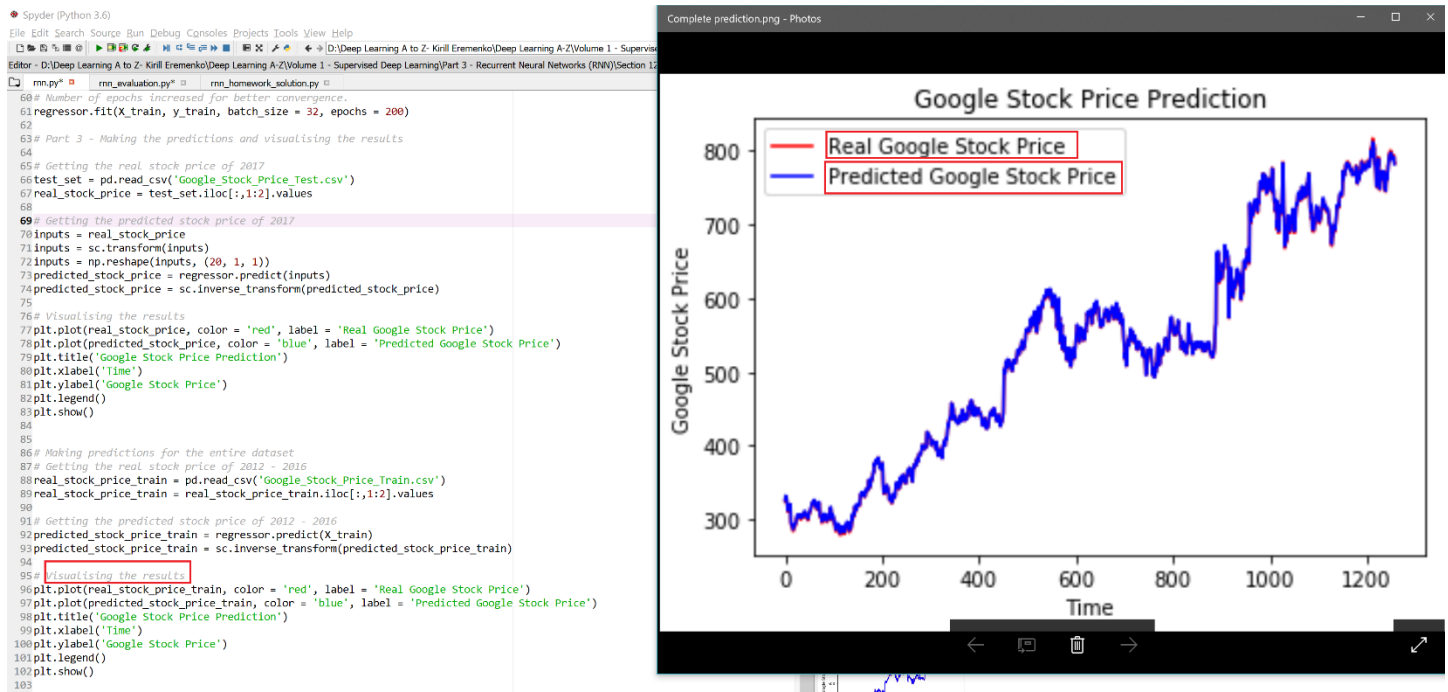
Python console

```

...: plt.xlabel('Time')
...: plt.ylabel('Google Stock Price')
...: plt.legend()
...: plt.show()

```

We compare our predictions with the actual prices through visualisation. At first, we are not able to see the real stock price. But after zooming in, we can see that the LSTM has very accurately predicted the prices and the Blue graph is mostly overlapping the Red one.



## Evaluating the RNN

RNNs are evaluated using the Root Mean Square (RMSE) of the test set. We will use the root using the math library. The mean squared error function is taken from the scikit-learn's metrics module. These 2 functions when combined, will calculate the rmse for the test sets actual values and the predictions we made. The actual RMSE value does not tell us anything about the size of the test set and how the error correlates to it. We need this in percentage. 800 is the average value of the stock in the test set, so we will divide the RMSE value by 800 to get a percentage value.

We see that the value is close to 0.4% which is a very low error rate and good for our predictions.

## Summary

Although the LSTM we have designed predicts quite accurately the stock prices of Google, the reason it is so accurate is because it is learning at time-step of 1. This leads to a reset of hidden layer, and this process goes on and the model is not learning anything useful. This output is not relevant because of this 1 time-step learning.

To make improvements in our model, we need to increase the time-step.

## Clarification

The project discussed previously is indeed from Stanford, but it is created by undergraduate students of the CS229 course. The students of that course submitted a final project for evaluation; it was just put online but it is not published anywhere. There is a big difference between reading a Stanford paper published online and reading an assignment written by a group of undergrads.