

# Predicting Stock/Crypto Prices using Deep Recurrent Neural Networks (RNN)

Long Short Term Memory (LSTM)  
Approach

Mohamed Ameer KHELIFA

University of California, Berkeley MEng Student



# 1 Introduction

This paper explains more in-depth the LSTM approach used in an effort to predict closing Stock and/or Cryptocurrency prices. It should be noted that the code written is relatively baseline and I doubt profits can be made solely following this program. As a matter of fact, prices hugely depend on a plethora of factors. To mention a few: the general sentiment (fear and greed index <https://alternative.me/crypto/fear-and-greed-index/>), trends (Bear vs Bull), elections...

Consequently, the purpose of this program is mainly to showcase that, with a relatively low computing power, and widely available data, one can build a program that can quite accurately follow the course of the market.

The core idea behind this approach is to use the past 60-day data as the  $X_{\text{train}}$  (entry data) and the following day (the 61<sup>st</sup> day) as the  $Y_{\text{train}}$  (prediction).

## 2 Long Short Term Memory (LSTM)

Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture[1] used in the field of deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections.

A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell.

LSTM networks are well-suited to classifying, processing, and making predictions based on time series data, since there can be lags of unknown duration between important events in a time series.

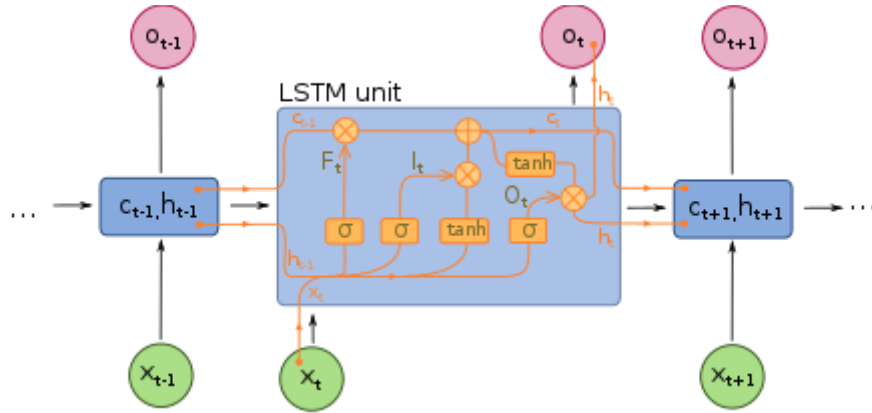


Fig-1: LSTM unit architecture

### 3 Predicting Closing Stock prices:

#### 3.1 Scraping Data

Before I begin building the neural network, I naturally must gather the data used to make predictions. As a result, I use pandas' data reader function to retrieve the data from YAHOO finance. Available stock data ranges from 2012-01-03 to today.

▶

#Get the stock values

df=scrapper.DataReader('AAPL',data\_source='yahoo', start='2012-01-01', end='2020-09-17')

#Show the data

df

High

Low

Open

Close

Volume

Adj Close

Date

2012-01-03

14.732142

14.607142

14.621428

14.686786

302220800.0

12.691425

2012-01-04

14.810000

14.617143

14.642858

14.765715

260022000.0

12.759631

2012-01-05

14.948215

14.738214

14.819643

14.929643

271269600.0

12.901293

2012-01-06

15.098214

14.972143

14.991786

15.085714

318292800.0

13.036158

2012-01-09

15.276786

15.048214

15.196428

15.061786

394024400.0

13.015480

...

...

...

...

...

...

...

2020-09-11

115.230003

110.000000

114.570000

112.000000

180860300.0

112.000000

2020-09-14

115.930000

112.800003

114.720001

115.360001

140150100.0

115.360001

2020-09-15

118.830002

113.610001

118.330002

115.540001

184642000.0

115.540001

2020-09-16

116.000000

112.040001

115.230003

112.129997

154679000.0

112.129997

2020-09-17

112.199997

108.709999

109.720001

110.339996

177637900.0

110.339996

2192 rows × 6 columns

Fig-2: Data available on the AAPL stock (APPLE Inc,)

Plotting the closing price shows the following graph:



Fig-3: Closing price history for the AAPL stock (APPLE Inc,)

## 3.2 Cleaning the Data

### 3.2.1 Creating new data frame

Create a new data frame with only the closing price and convert it to an array.


Then create a variable to store the length of the training data set. I want the training data set to contain about 80% of the data.

```
#Create new data frame with only the Close column
data=df.filter(['Close'])
#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
```

Fig-4: New data frame

### 3.2.2 Scaling the data and creating the training sets

It is generally good practice to scale your data before giving it to the neural network. The data set to be values between 0 and 1 inclusive.

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

scaled_data



 array([[0.0061488 ],
        [0.00680527],
        [0.00816869],
        ...,
        [0.8449671 ],
        [0.81660535],
        [0.80171752]])
```

Fig-5: Scaled data

Now, we create the training data set. We use 80% of the total data.

```
 #Create 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(60, len(train_data)):
    x_train.append(train_data[i-60:i,0])
    y_train.append(train_data[i,0])


[9] #convert the x_train and y_train to numpy arrays
x_train, y_train=np.array(x_train),np.array(y_train)

[10] #Reshape the data
x_train=np.reshape(x_train,(x_train.shape[0],x_train.shape[1],1))
x_train.shape
```

Fig-6: Training data set

### 3.3 Building and training the RNN

We choose to implement a deep neural network composed of two LSTM layers with 50 neurons and two Dense layers, one with 25 neurons and the other with 1 neuron. This is standard practice in LSTM literature.

```
 #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))

[12] #Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

[13] #Train the model
model.fit(x_train, y_train, batch_size=1, epochs=1)


 1694/1694 [=====] - 43s 26ms/step - loss: 2.6019e-04
<tensorflow.python.keras.callbacks.History at 0x7f40dbf39668>
```

Fig-6: Building the model

We note that training the model was relatively fast (43s) with a 26ms/step. Consequently, it can be improved by adding new neurons. However, further testing is mandatory because a risk of overfitting is existing when adding too many layers.

### 3.4 Results

#### 3.4.1 Plotting the results

Plotting the predictions, we note that they follow quite accurately the validation data (real closing prices). This shows that this modest model with limited resources is able to yield good results which substantiates the relevance of RNNs and more precisely LSTM networks compared to other types of deep neural networks in the study of times series.

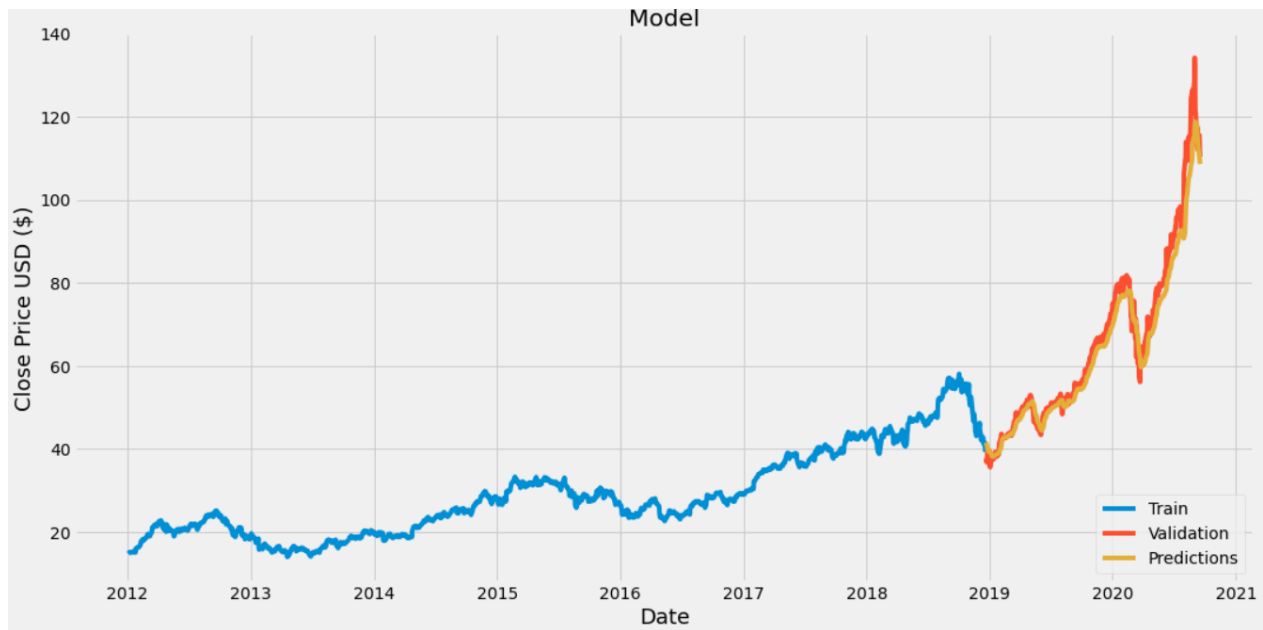


Fig-7: Model Performance

### 3.4.2 RMSE

In order to have a more precise idea of the performance of the model, it is good practice to calculate the Root Mean Squared Error (RMSE).

The RMSE of predicted values  $\hat{y}_t$  for times  $t$  of a regression's dependent variable  $y_t$  with variables observed over  $T$  times, is computed for  $T$  different predictions as the square root of the mean of the squares of the deviations:

$$\text{RMSE} = \sqrt{\frac{\sum_{t=1}^T (\hat{y}_t - y_t)^2}{T}}.$$

```
[18] #Evaluate the model
      #Get the root mean squared error (RMSE)
      rmse=np.sqrt(np.mean(((predictions- y_test)**2)))
      rmse
```

4.209380927583289

A RMSE value of  $\sim 4.2$  is good compared to the scale of our stock closing prices ranging from 14 to 110.



## 4 Predicting Cryptocurrency prices

### 4.1 Bitcoin

Bitcoin is a digital currency created in January 2009 following the housing market crash. Bitcoin is a collection of computers, or nodes, that all run Bitcoin's code and store its blockchain. A blockchain can be thought of as a collection of blocks. In each block is a collection of transactions. Because all these computers running the blockchain have the same list of blocks and transactions and can transparently see these new blocks being filled with new Bitcoin transactions, no one can cheat the system which makes it interesting compared to FIAT currencies.

### 4.2 Model and Results

We use the same approach to try and predict the closing prices of the BTC-USD pair. BTC stands for Bitcoin.

What makes this stock interesting to study is that Cryptocurrencies in general and Bitcoin in particular are known to be highly volatile (for instance, it slumped to below \$11,500 at one point on Friday – touching \$11,159 – having started the week at a record high close to \$20,000 and in its biggest weekly fall since 2013).

Plotting the closing price history of the BTC-USD pair we get:



Fig-8: Closing price history BTC-USD

Using the previously trained neural network, we can calculate the model in comparison with the real price values. We obtain:

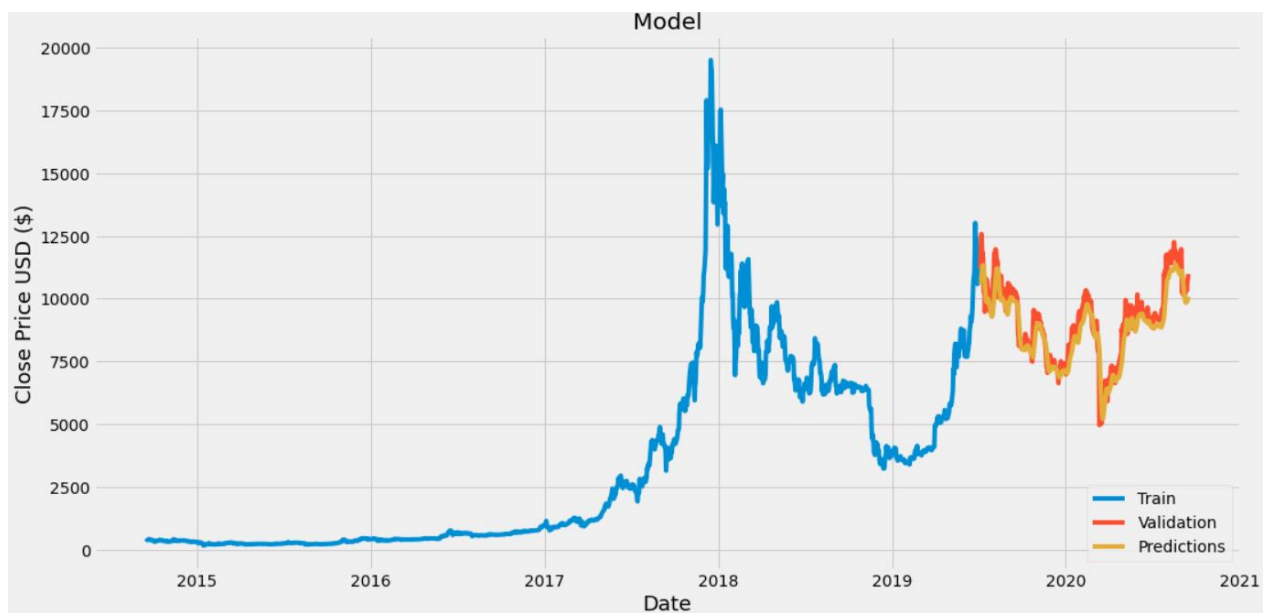


Fig-9: Model vs Validation

It is noted that, despite the high volatility of the crypto market, the model follows relatively well the fluctuations of the price. It should be mentioned however that, in the case of BTC, due to the already high prices of 1 BTC (10,000\$), the difference between the model and the validation can get as high as 900\$. This substantiates even more the fact that using this program to day trade is not sustainable. It must rather be viewed as a way to study the general tendencies of the market.

## 5 Conclusion

First of all, I would like to thank all deep learning enthusiasts for the quality of their open sourced work. Indeed, using LSTM to predict stock prices is a widely known approach and as result the works available on the subject were a great source of inspiration to me.

As mentioned above, stock/crypto prices are influenced by many factors and solely basing predictions on past prices can only be lackluster.

To improve the model, one can try to implement variations of this algorithm adding aspects of Technical Analysis (using indicators such as MACD, RSI...) as well as Fundamental Analysis (Fear/Greed index, market trends, socioeconomic indicators...).