Forex Prediction with LSTM Deep Learning

Coursera Advance Data Science Capstone Project

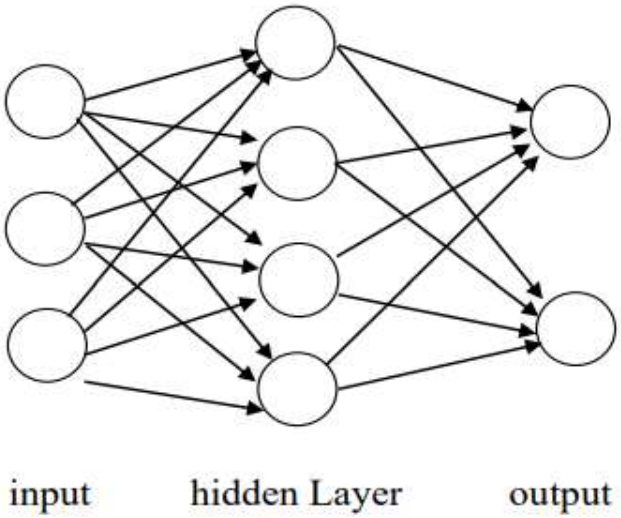
by Ben Sung

August 2019

Description

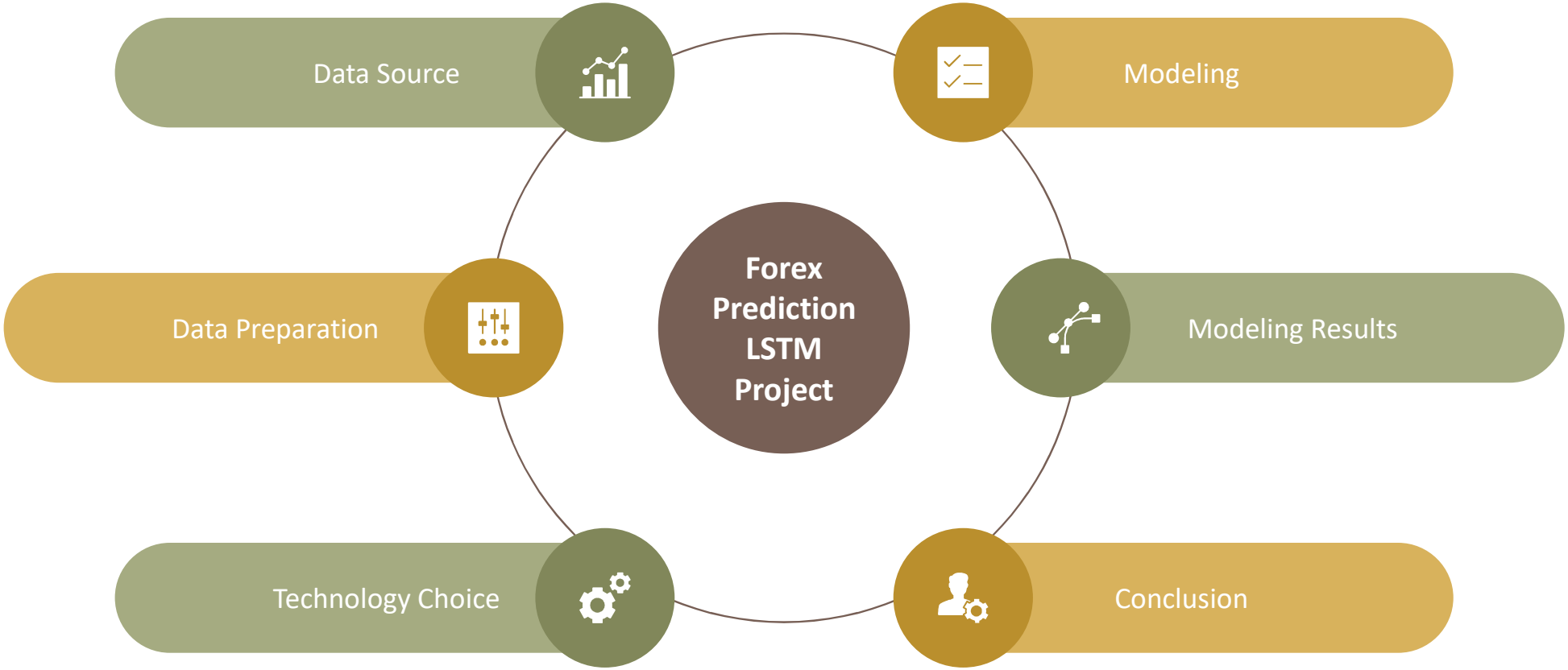
- Forex or Foreign Exchange or FX trading is the conversion of one currency with another. It is the most traded markets in the world with daily trading of 5 trillion dollars on average.
- Traders have used wide variety of strategies and methodologies to improve on their prediction and profits
- With the emerging popularity of machine learning and deep learning, traders have been applying these approaches for trading decisions.
- In this project, we will use LSTM (Long Short-Term Memory) neural network to determine the accuracy of predicting Forex prices with most popular Euro vs USD exchange rate as example.

Project Goals



- Predicting foreign exchange rates for trading purpose
- Implementing machine learning algorithm using deep learning model on time series information – LSTM (Long Short-Term Memory)
- Determine the prediction accuracy if LSTM is suitable for Forex trader
- The Model **performance indicator** used for this projects are mean absolute error (MAE) and root mean square error (RMSE) to detect the accuracy of the prediction to the actual values

Project Analysis



DATA SOURCE

Data Source

Forex EURUSD from Jan 2011 to Dec 2018 (8 years) downloaded from QuanDL

```
# Data source - fetch dataframe for Forex EURUSD from Jan 2011 to Dec 2018 (8 years)
df = pd.read_csv('EURUSD_DAY_2011_2018.csv')
df.columns = ('Date', 'Time', 'Open', 'High', 'Low', 'Close', 'Volume')
print (df.head())
print (df.shape)
```

```
         Date   Time     Open     High      Low    Close  Volume
0  2011.01.03  00:00  1.33560  1.33947  1.32500  1.33551  164193
1  2011.01.04  00:00  1.33566  1.34298  1.32923  1.33179  215600
2  2011.01.05  00:00  1.33181  1.33248  1.31262  1.31556  204024
3  2011.01.06  00:00  1.31556  1.31698  1.29723  1.29787  185391
4  2011.01.07  00:00  1.29787  1.30209  1.29045  1.29060  180709
(2501, 7)
```

Data Source

- Downloaded from QuanDL for Forex prices for EURO vs USD from Jan 2011 to Dec 2018.
- Consists of 2501 rows (days) with 7 columns: Date, Time, Open, High, Low, Close and Volume

DATA PREPARATION

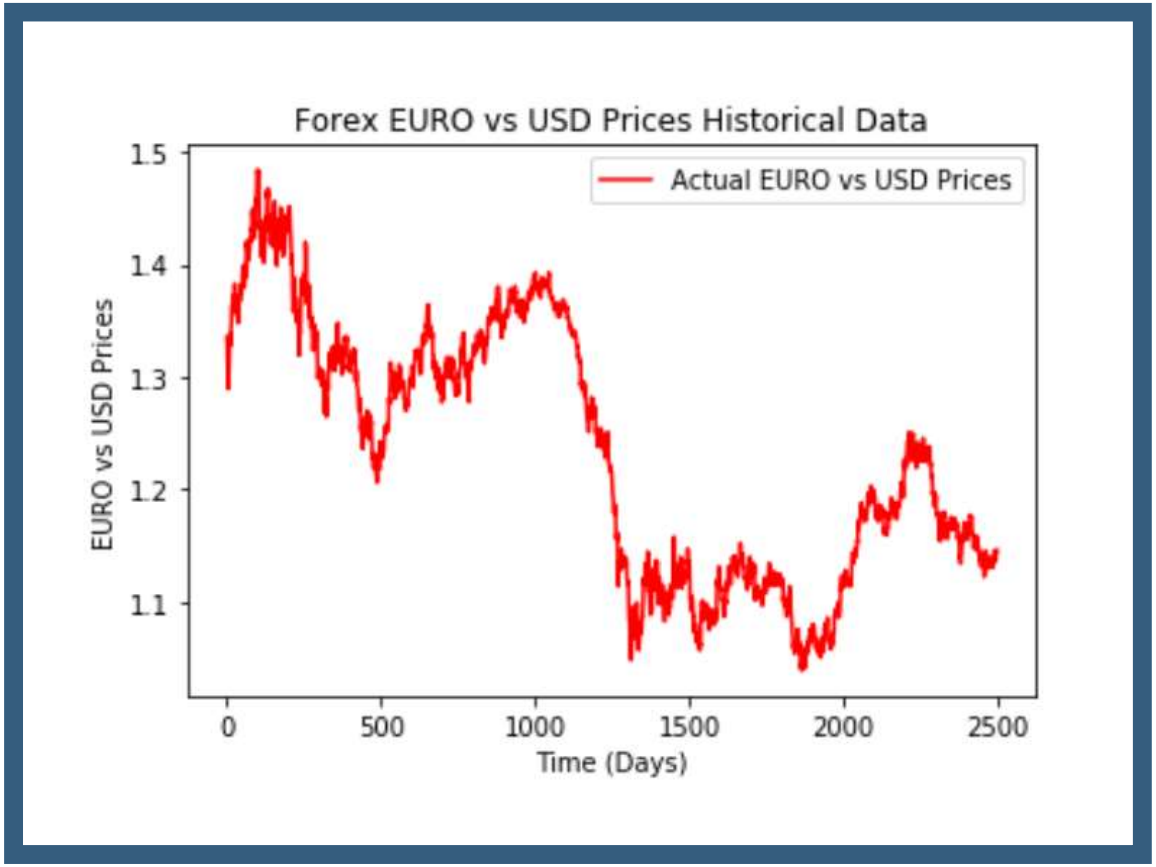
```
# Drop unused columns/features/dimensions
df.drop(columns=['Time', 'Open', 'High', 'Low', 'Volume'], inplace=True)
print(df.head())
print(df.shape)
```

```
         Date    Close
0  2011.01.03  1.33551
1  2011.01.04  1.33179
2  2011.01.05  1.31556
3  2011.01.06  1.29787
4  2011.01.07  1.29060
(2501, 2)
```

Data Cleaning

- I choose the previous Closed prices to predict the future prices, as they indicate the closed price of the day.
- Therefore for **feature engineering**, I drop the unrelated columns: Time, Open, High, Low and Volume.
- No other cleaning on the data necessary as there are no missing or error data.

DATA PREPARATION



Plot of the data

- 2501 days of actual Euro vs USD prices from Jan 2011 to Dec 2018

MODELING

Technology

- Framework chosen for this project:
  - Python with pandas, numpy, matplotlib, keras and h5py libraries
- Algorithm that I choose is LSTM from Keras using TensorFlow backend
  - Stateful deep learning algorithm suitable for time series prediction
  - Neural Network of 4 layers (Input, 2 LSTM layers with 10 neurons each and a Dense output layer)
- Jupyter Lab
  - Easy collaboration, streamline and enable more productivity for data scientist projects

MODELING

Stateful LSTM from Keras using TensorFlow backend

- Stateful is used as the previous Forex prices (historical data) would influence the future prices. Sequences in batches are related to each other.
- Batch-size of 64, Epochs of 120 and Timesteps of 30 will be used to run the LSTM neural network layers to make the predictions

```
# defining the batch size and number of epochs
batch_size = 64
epochs = 120
timesteps = 30 # 30 steps to the left or we are looking at sliding windows of 10 days in the past to predict
print("Batch size: ", batch_size)
print("Epochs: ", epochs)
print("Timesteps (sliding windows): ", timesteps)
```

```
Batch size:  64
Epochs:  120
Timesteps (sliding windows):  30
```

MODELING

```
# Pre-processing - Construct our Input and Output Data Construction for LSTM
# First, increase the upper boundary of traning set size by mulipling by two
# Adding timesteps * 2
upper_train = length + timesteps*2
df_data_1_train = df[0:upper_train]
training_set = df_data_1_train.iloc[:,1:2].values
print('Training Set: ', training_set)
print('Training Shape: ', training_set.shape)
```

```
Training Set:  [[1.33551]
 [1.33179]
 [1.31556]
 ...
 [1.18671]
 [1.18467]
 [1.19137]]
Training Shape:  (2300, 1)
```

```
# Feature Scaling
# scale between 0 and 1. the weights are esier to find.
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))
training_set_scaled = sc.fit_transform(np.float64(training_set))
training_set_scaled.shape
```

```
(2300, 1)
```

Constructing Input and Output Data for LSTM

- First, increase the upper boundary of training set size by multiplying by two
- The training shape will be 2300 records
- Apply Feature scaling between 0 to 1 to fit LSTM

MODELING

```
# Input data - X_train (timesteps = 10)
X_train = []
# Ouput data - y_train - predicted future values
y_train = []

# Creating a data structure with n timesteps
print("Length + timesteps: ", length + timesteps)
for i in range(timesteps, length + timesteps):
    X_train.append(training_set_scaled[i-timesteps:i,0])
    y_train.append(training_set_scaled[i:i+timesteps,0])

print("Length of X_train", len(X_train))
print("Lenght of y_train", len(y_train))

# create X_train matrix
# 30 items per array (timestep)
print("X_train 3D Matrix: ", X_train[0:2])
print("X_train array shape: ", np.array(X_train).shape)

#create Y_train matrix
#30 items per array (timestep)
print("y_train 3D Matrix: ", y_train[0:2])
print("y_train array shape: ", np.array(y_train).shape)
```

```
Length + timesteps:  2270
Length of X_train 2240
Lenght of y_train 2240
X_train 3D Matrix:  [array([0.66623966, 0.65787936, 0.62140417, 0.58164779, 0.56530924,
       0.56261237, 0.5782767 , 0.58209727, 0.61477436, 0.66522834,
       0.6733639 , 0.67147609, 0.64956401, 0.6712963 , 0.68790453,
       0.69093851, 0.725863  , 0.72548094, 0.73078479, 0.74029126,
       0.74593222, 0.74860662, 0.72350324, 0.71487325, 0.74678623,
       0.77256383, 0.76975458, 0.72649227, 0.71673858, 0.71514293]), array([0.65787936,
7,
       0.5782767 , 0.58209727, 0.61477436, 0.66522834, 0.6733639 ,
       0.67147609, 0.64956401, 0.6712963 , 0.68790453, 0.69093851,
       0.725863  , 0.72548094, 0.73078479, 0.74029126, 0.74593222,
       0.74860662, 0.72350324, 0.71487325, 0.74678623, 0.77256383,
       0.76975458, 0.72649227, 0.71673858, 0.71514293, 0.71952535])]
X_train array shape:  (2240, 30)
y_train 3D Matrix:  [array([0.71952535, 0.72687433, 0.74799982, 0.72190759, 0.71013125,
       0.70141136, 0.69403991, 0.69822006, 0.71530025, 0.72629   ,
       0.74197681, 0.7463817 , 0.73945973, 0.73739213, 0.75721413,
       0.77054117, 0.75564096, 0.75119112, 0.77076591, 0.7589671 ,
       0.78081176, 0.80119561, 0.80809511, 0.80890417, 0.80263394,
       0.78739662, 0.78941927, 0.7662936 , 0.78923948, 0.79665588]), array([0.72687433,
6,
       0.69403991, 0.69822006, 0.71530025, 0.72629   , 0.74197681,
       0.7463817 , 0.73945973, 0.73739213, 0.75721413, 0.77054117,
       0.75564096, 0.75119112, 0.77076591, 0.7589671 , 0.78081176,
       0.80119561, 0.80809511, 0.80890417, 0.80263394, 0.78739662,
       0.78941927, 0.7662936 , 0.78923948, 0.79665588, 0.80894912])]
y_train array shape:  (2240, 30)
```

Create a data structure with n timesteps

- Create X_train and y_train matrix of 30 items per arrays to be consumed by LSTM neural network layers in later sections
- The results will be 2 arrays with shape of 2240 (records) by 30 (timesteps)

MODELING

```
########## Building the LSTM
# Importing the Keras libraries and packages

from keras.layers import Dense
from keras.layers import Input, LSTM
from keras.models import Model
import h5py
```

```
########## Anatomy of a LSTM Node
# Initialising the LSTM Model with MAE Loss-Function
# Keras has 2 API: Sequential API and Functional API
# Here, we use Functional API

# Input layer:
inputs_1_mae = Input(batch_shape=(batch_size,timesteps,1))

# Each layer is the input of the next layer
# mae -> mean absolute error loss function
# We use 10 nodes
lstm_1_mae = LSTM(10, stateful=True, return_sequences=True)(inputs_1_mae)
lstm_2_mae = LSTM(10, stateful=True, return_sequences=True)(lstm_1_mae)

# Output layer
output_1_mae = Dense(units = 1)(lstm_2_mae)

regressor_mae = Model(inputs=inputs_1_mae, outputs = output_1_mae)

# Optimizer - adam is fast starting off and then gets slower and more precise
regressor_mae.compile(optimizer='adam', loss = 'mae')
regressor_mae.summary()
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (64, 30, 1)               0
_________________________________________________________________
lstm_1 (LSTM)                (64, 30, 10)              480
_________________________________________________________________
lstm_2 (LSTM)                (64, 30, 10)              840
_________________________________________________________________
dense_1 (Dense)              (64, 30, 1)               11
=================================================================
Total params: 1,331
Trainable params: 1,331
Non-trainable params: 0
_________________________________________________________________
```

Building the LSTM with MAE

- First, we need to imports all the keras libraries and packages
- Then, we initialize the LSTM model with MAE Loss-Function
- For Keras in this case, we use Functional API, instead of Sequential API for flexibility to create the layers
- We create 3 layers: Input, 2 x LSTM (with 10 nodes) and output layers
- Using ‘adam’ optimizer, we compile our layers

Note: MAE – Mean Absolute Error

MODELING

```
########## Model Training with Statefull
# Fitting the training set

#Statefull
for i in range(epochs):
    print("Epoch: " + str(i))
    #run through all data but the cell, hidden state are used for the next batch.
    regressor_mae.fit(X_train, y_train, shuffle=False, epochs = 1, batch_size = batch_size)
    #resets only the states but the weights, cell and hidden are kept.
    regressor_mae.reset_states()

#Stateless
#between the batches the cell and hidden states are lost.
#regressor_mae.fit(X_train, y_train, shuffle=False, epochs = epochs, batch_size = batch_size)
```

```
Epoch: 116
Epoch 1/1
2240/2240 [==============================] - 0s 133us/step - loss: 0.0599
Epoch: 117
Epoch 1/1
2240/2240 [==============================] - 0s 137us/step - loss: 0.0600
Epoch: 118
Epoch 1/1
2240/2240 [==============================] - 0s 140us/step - loss: 0.0608
Epoch: 119
Epoch 1/1
2240/2240 [==============================] - 0s 138us/step - loss: 0.0644
```

Model Training with Stateful LSTM

- We run through all data in our X_train and y_train with shuffle=False for Stateful LSTM
- Re-run for the number of epochs of 120
- At end of each iteration, we reset the states but the weights, cell and hidden are kept

TESTING

```
def get_test_length(dataset, batch_size):
    test_length_values = []
    for x in range(len(dataset) - 200, len(dataset) - timesteps*2):
        modulo=(x-upper_train)%batch_size
        if (modulo == 0):
            test_length_values.append(x)
            print(x)
    return (max(test_length_values))
```

```
test_length = get_test_length(df, batch_size)
print("Test Length: ", test_length)

upper_test = test_length + timesteps*2
testset_length = test_length - upper_train
print("Testset Length: ", testset_length)
print("Upper Train, Upper Test, Length: ", upper_train, upper_test, len(df))
```

```
2364
2428
Test Length:  2428
Testset Length:  128
Upper Train, Upper Test, Length:  2300 2488 2501
```

Preparing for test data

- Calculate the test, Testset and Upper Train data
- Test data size is 128 records; therefore from records [2300..2428] is used for our prediction test!

TESTING

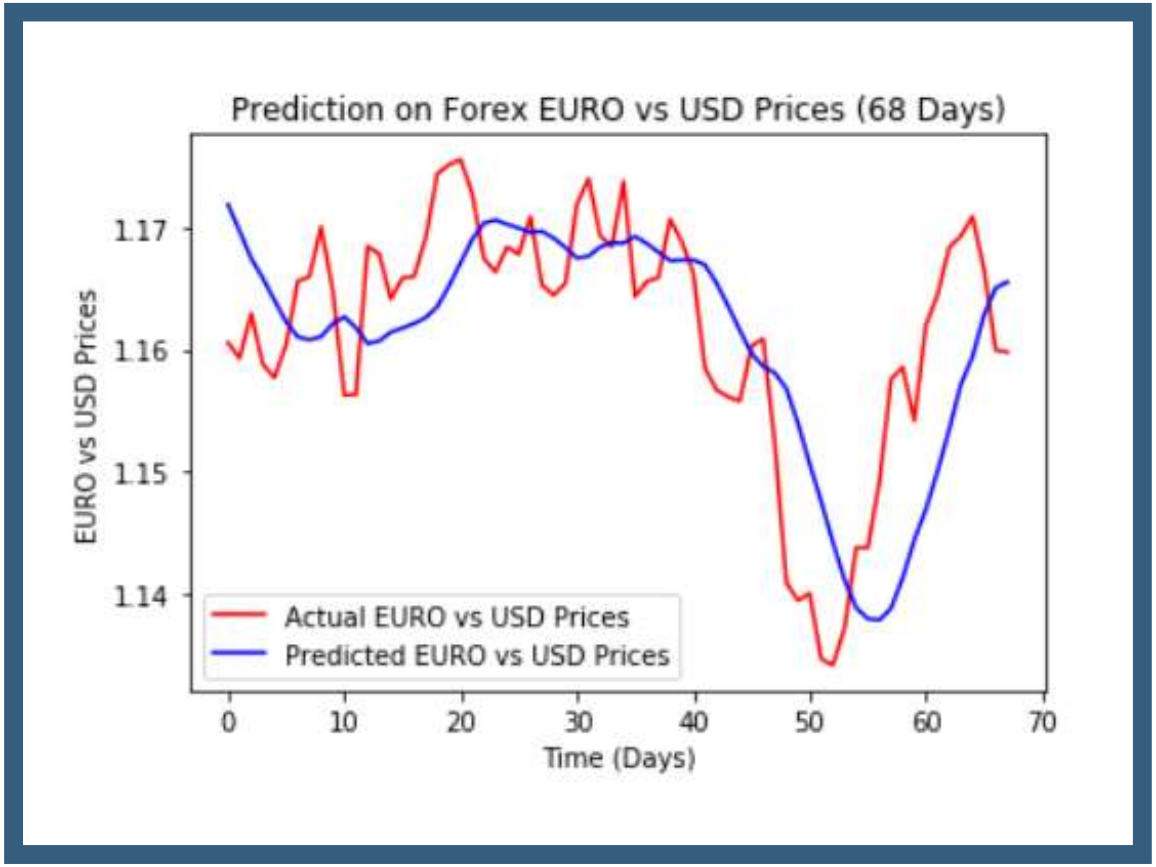
```
# prediction
predicted_bcg_values_test_mae = regressor_mae.predict(X_test, batch_size=batch_size)
regressor_mae.reset_states()

print(predicted_bcg_values_test_mae.shape)

# reshaping
predicted_bcg_values_test_mae = np.reshape(predicted_bcg_values_test_mae,
                                           (predicted_bcg_values_test_mae.shape[0],
                                            predicted_bcg_values_test_mae.shape[1]))

print(predicted_bcg_values_test_mae.shape)

# inverse transform for forecasting
predicted_bcg_values_test_mae = sc.inverse_transform(predicted_bcg_values_test_mae)

# creating y_test data
y_test = []
for j in range(0, testset_length - timesteps):
    y_test = np.append(y_test, predicted_bcg_values_test_mae[j, timesteps-1])

# reshaping
y_test = np.reshape(y_test, (y_test.shape[0], 1))

print(y_test.shape)
```

```
(128, 30, 1)
(128, 30)
(98, 1)
```

Run Prediction on the 128 test data

- We will reshape and inverse transform for forecasting the test data to get 98 records of predicted test data.
- y_test contains the 98 predicted data.

TESTING



Testing Result (MAE)

- Plot of the 68 predicted EURO vs USD prices compares with actual prices.
- Test data of 128 – 30 timesteps = 68 predicted data points

TESTING

```python
#MSE (mean sqared error)
import math
from sklearn.metrics import mean_squared_error
rmse = math.sqrt(mean_squared_error(test_set[timesteps:len(y_test)], y_test[0:len(y_test) - timesteps]))
print(r"RMSE (Root Mean Square Error):", rmse)
```

```
RMSE (Root Mean Square Error): 0.009613904012023765
```

```python
#MAE (mean absolut error)
from sklearn.metrics import mean_absolute_error
mae = mean_absolute_error(test_set[timesteps:len(y_test)], y_test[0:len(y_test) - timesteps])
print(r"MAE (Mean Absolute Error)", mae)
```

```
MAE (Mean Absolute Error) 0.007628275272705977
```

Checking the accuracy of prediction with MAE

- Test results:
  - RMSE: 0.00961
  - MAE : 0.00763

TESTING

```
# Initialising the LSTM Model with MSE Loss Function

inputs_1_mse = Input(batch_shape=(batch_size,timesteps,1))
lstm_1_mse = LSTM(10, stateful=True, return_sequences=True)(inputs_1_mse)
lstm_2_mse = LSTM(10, stateful=True, return_sequences=True)(lstm_1_mse)

output_1_mse = Dense(units = 1)(lstm_2_mse)

regressor_mse = Model(inputs=inputs_1_mse, outputs = output_1_mse)

#mse -> mean squared error as loss function
regressor_mse.compile(optimizer='adam', loss = 'mse')
regressor_mse.summary()
```

```
Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (64, 30, 1)               0
_________________________________________________________________
lstm_3 (LSTM)                (64, 30, 10)              480
_________________________________________________________________
lstm_4 (LSTM)                (64, 30, 10)              840
_________________________________________________________________
dense_2 (Dense)              (64, 30, 1)               11
=================================================================
Total params: 1,331
Trainable params: 1,331
Non-trainable params: 0
_________________________________________________________________
```

Re-test with LSTM using MSE for loss

- Set loss to MSE and compile the regressor

TESTING

```
#MSE (mean sqared error)
import math
from sklearn.metrics import mean_squared_error
rmse = math.sqrt(mean_squared_error(test_set[timesteps:len(y_test)], y_test[0:len(y_test) - timesteps]))
print(r"RMSE (Root Mean Square Error):", rmse)
```

```
RMSE (Root Mean Square Error): 0.00875689492891446
```

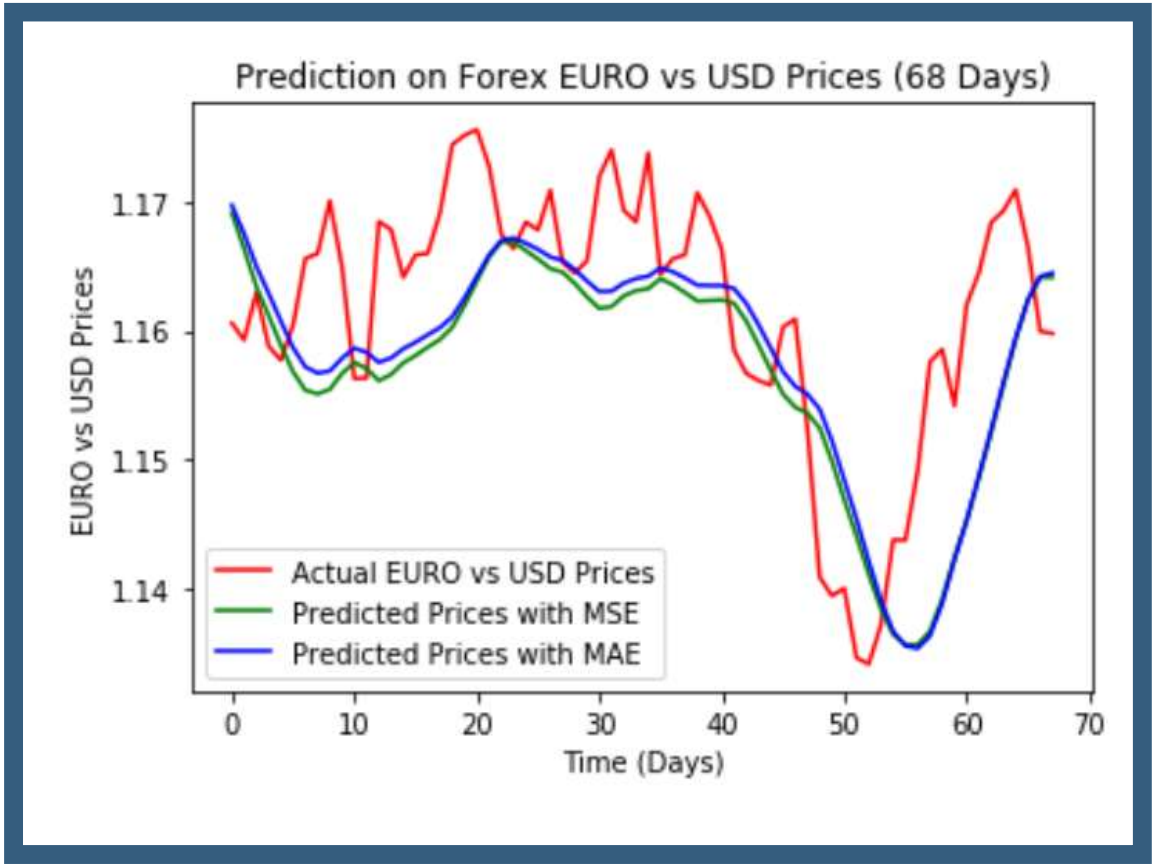
```
#MAE (mean absolut error)
from sklearn.metrics import mean_absolute_error
mae = mean_absolute_error(test_set[timesteps:len(y_test)], y_test[0:len(y_test) - timesteps])
print(r"MAE (Mean Absolute Error)", mae)
```

```
MAE (Mean Absolute Error) 0.007129522263583008
```

Checking the accuracy of prediction MAE vs MSE

- Test results (MAE):
  - RMSE: 0.00961
  - MAE : 0.00763
- Test results (MSE):
  - RMSE: 0.00797
  - MAE : 0.00622

TESTING



Testing Result using MSE vs MAE

- Plot of the 68 predicted EURO vs USD prices compares with actual prices.
- Test data of 128 – 30 timesteps = 68 predicted data points
- From the plot we also observe that prediction using MSE as loss is similar with prediction using MAE as loss

Conclusion

- In this project, we use LSTM to predict Forex prices of Euro/USD exchange rate. The predicted values looks promising with low mean absolute error (MAE) around 0.00622 to 0.00763 and root mean square error (RMSE) around 0.00797 to 0.00961. This is due to the fact that the prediction is done one day at a time, so we add the actual value to the test data on predicting the next day; therefore reducing the error.

- From the graph of predicted vs actual prices, the predicted prices seems lagging in timing behind the actual prices. This would be a concern for trader, as the reaction to sell or buy would be slow and would cause losses.

- Further fine tune of the LSTM parameters, epochs and timesteps; add layers and neurons; as well as data cleaning on the type of days (holidays, weekends, month ends, etc.) may improve on the accuracy of the prediction.

- As the prediction is based solely on the previous data on single currency; therefore, other currencies data, other type of trading and even external factors such as expert/domain knowledges using technical indicator, social/news sentiments, political events and others might be added as relevant features to be considered to improve the prediction.

Related Documents

- Coursera Crude Oil Price Prediction
  - https://github.com/romeokienzler/developerWorks/blob/master/coursera/ai/week3/lstm_crude_oil_price_prediction.ipynb
- Jupyter Lab – Source Code
  - https://github.com/b3n12/DeepLearning/blob/master/CourseraForexLSTM.ipynb
- Data for Euro vs USD exchange rate prices (2011-2018)
  - https://github.com/b3n12/DeepLearning/blob/master/EURUSD_DAY_2011_2018.csv

Thank You