CSYE 7245 - Big-Data Systems and Intelligence Analytics

Assignment 1

Bitcoin Historical Data

Amogh Chakkarwar | chakkarwar.a@husky.neu.edu

Abstract-- Bitcoin has recently attracted considerable attention in the fields of economics, cryptography, and computer science due to its inherent nature of combining encryption technology and monetary units. Bitcoin is a cryptocurrency and worldwide payment system. It is the first decentralized digital currency, as the system works without a central bank or single administrator. The network is peer-to-peer and transactions take place between users directly, without an intermediary. These transactions are verified by network nodes through the use of cryptography and recorded in a public distributed ledger called a blockchain. Bitcoin was invented by an unknown person or group of people under the name Satoshi Nakamoto and released as open-source software in 2009.

1. Introduction

In this assignment I have implemented **RNN (Recurrent Neural Network)** using TensorFlow to predict the price of bitcoin using the dataset from Coinbase a Bitcoin company. Nowadays, Due to the sudden boom in Bitcoin prices everyone wants to invest into Bitcoins. Bitcoins are either "mined" by a computer through a process of solving increasingly complex mathematical algorithms or purchased with standard national money currencies and placed into a "Bitcoin wallet" that is accessed through a smartphone or computer. Predicting future Bitcoin prices is an interesting subject.

2. Methodology: Background

Traditional neural networks can’t allow information to persist, and it seems like a major shortcoming. Recurrent neural networks address this issue. They are networks with loops in them, allowing information to persist.

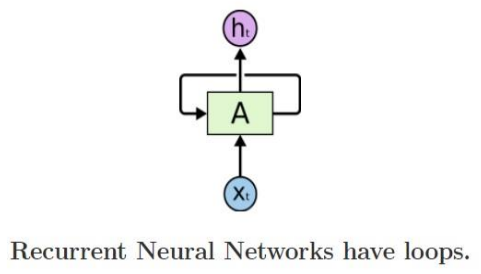
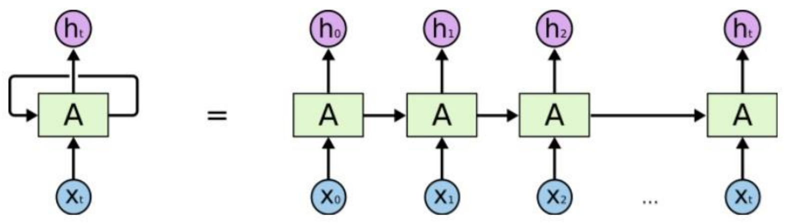


Fig. 1 : RNN Loop

In the above diagram Fig. 1, a chunk of neural network, AA, looks at some input xt and outputs a value ht. A loop allows information to be passed from one step of the network to the next.
A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor. Consider below as when we unroll the loop:



An unrolled recurrent neural network.

Fig. 2: RNN Unrolled

This chain-like nature reveals that recurrent neural networks are intimately related to sequences and lists. They're the natural architecture of neural network to use for such data.

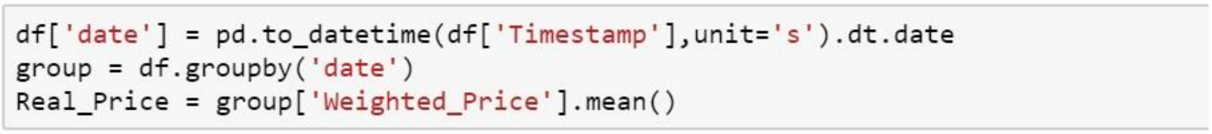
3. Dataset

The dataset for this assignment is taken from kaggel.com named as **Bitcoin Historical Data.** CSV files for select bitcoin exchanges for the time period of Dec 2014 to Jan 2018, with minute to minute updates of OHLC (Open, High, Low, Close), Volume in BTC and indicated currency, and weighted bitcoin price. Timestamps are in Unix time. This dataset has around 1.5 Million records.

4. Implementation

Step 1:

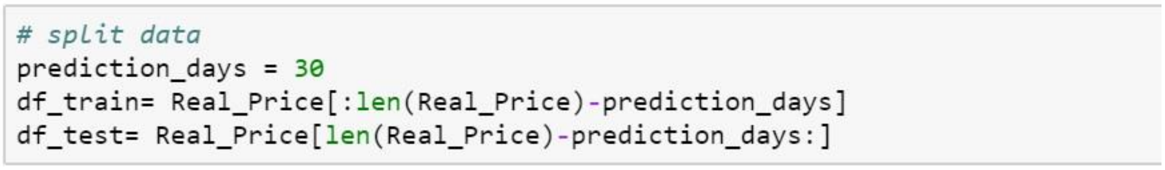
Read CSV file into the DataFrame and groupby dates. As the csv has records of each minute by minute values, I am taking mean value of Weighted_Price and assigning it to each day data in Real_Price.



```
df['date'] = pd.to_datetime(df['Timestamp'],unit='s').dt.date
group = df.groupby('date')
Real_Price = group['Weighted_Price'].mean()
```

Step: 2

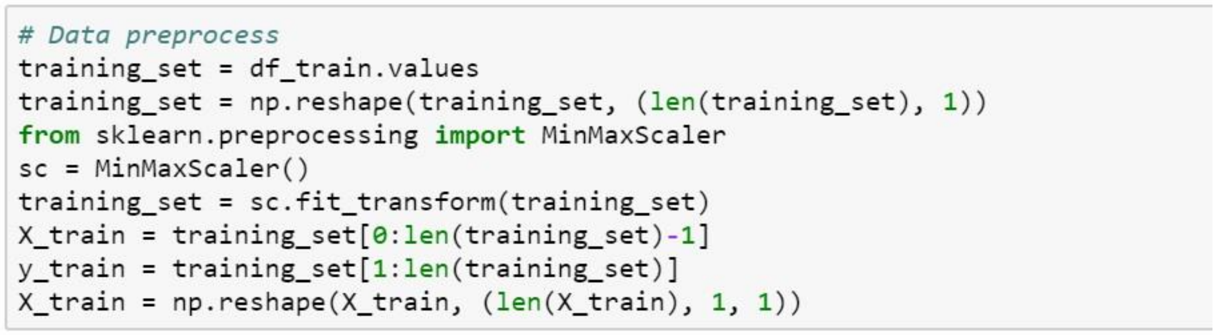
I am predicting for 30 days and 10 days.



```
# split data
prediction_days = 30
df_train= Real_Price[:len(Real_Price)-prediction_days]
df_test= Real_Price[len(Real_Price)-prediction_days:]
```

Step 3:

Using MinMaxScaler to scale the data and reshape as I am using Keras.

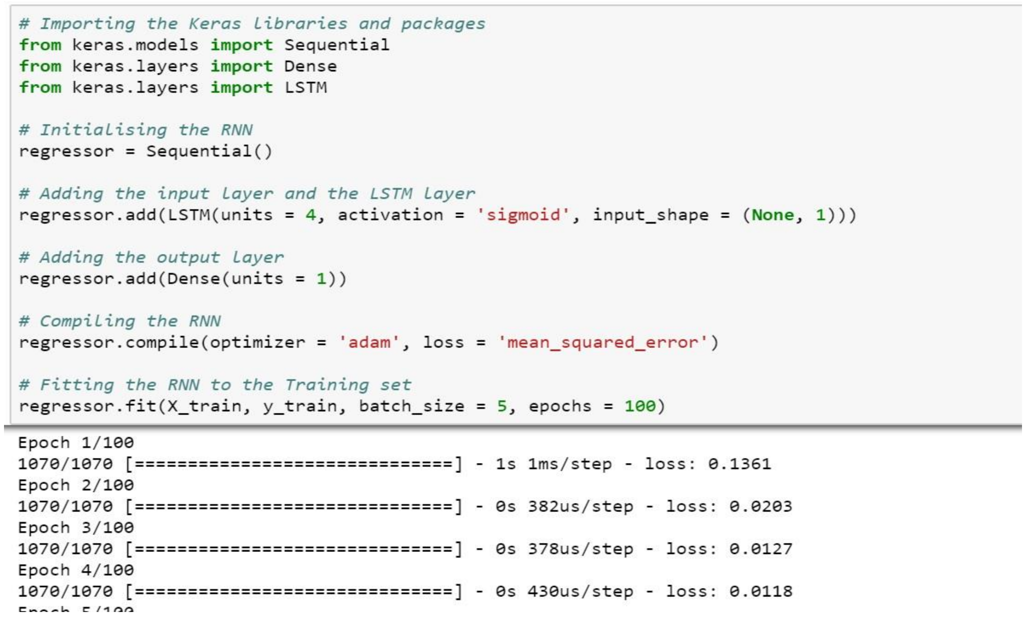


```
# Data preprocess
training_set = df_train.values
training_set = np.reshape(training_set, (len(training_set), 1))
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler()
training_set = sc.fit_transform(training_set)
X_train = training_set[0:len(training_set)-1]
y_train = training_set[1:len(training_set)]
X_train = np.reshape(X_train, (len(X_train), 1, 1))
```

Step 4:

Build the RNN model using Keras. I am taking epochs as 100 and batch size of 5.
One Epoch is when an ENTIRE dataset is passed forward and backward through the neural network only ONCE. Since, one epoch is too big to feed to the computer at once we divide it in several smaller batches.
One epoch leads to underfitting. As the number of epochs increases, more number of times the weight are changed in the neural network and the model goes from underfitting to optimal to overfitting.

Batch size is the total number of training examples present in single batch. You have to divide dataset into Number of Batches or sets or parts as you cannot pass entire dataset into the neural network at once.

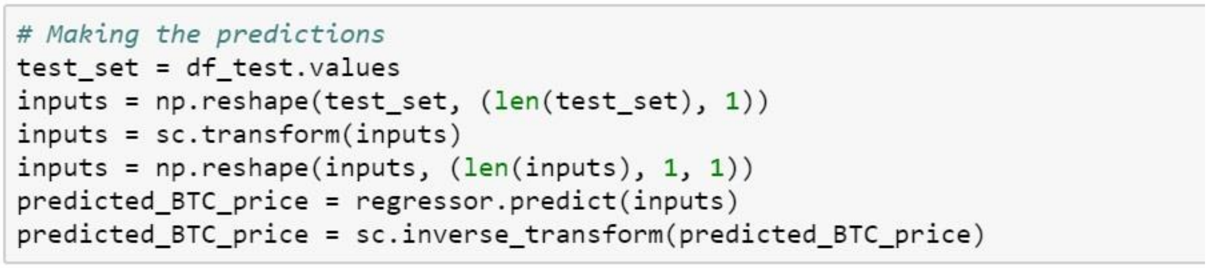


```
# Importing the Keras libraries and packages
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM

# Initialising the RNN
regressor = Sequential()

# Adding the input layer and the LSTM layer
regressor.add(LSTM(units = 4, activation = 'sigmoid', input_shape = (None, 1)))

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

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

# Fitting the RNN to the Training set
regressor.fit(X_train, y_train, batch_size = 5, epochs = 100)
```

```
Epoch 1/100
1070/1070 [==============================] - 1s 1ms/step - loss: 0.1361
Epoch 2/100
1070/1070 [==============================] - 0s 382us/step - loss: 0.0203
Epoch 3/100
1070/1070 [==============================] - 0s 378us/step - loss: 0.0127
Epoch 4/100
1070/1070 [==============================] - 0s 430us/step - loss: 0.0118
```

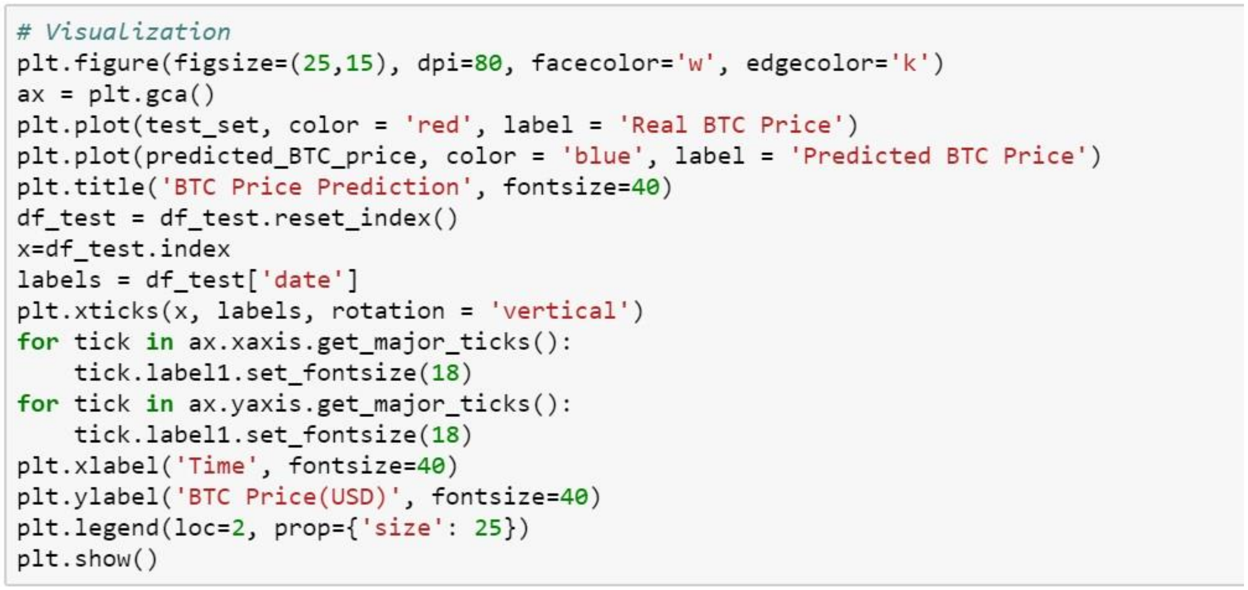
Step 5:

Applying predict function on RNN



```
# Making the predictions
test_set = df_test.values
inputs = np.reshape(test_set, (len(test_set), 1))
inputs = sc.transform(inputs)
inputs = np.reshape(inputs, (len(inputs), 1, 1))
predicted_BTC_price = regressor.predict(inputs)
predicted_BTC_price = sc.inverse_transform(predicted_BTC_price)
```

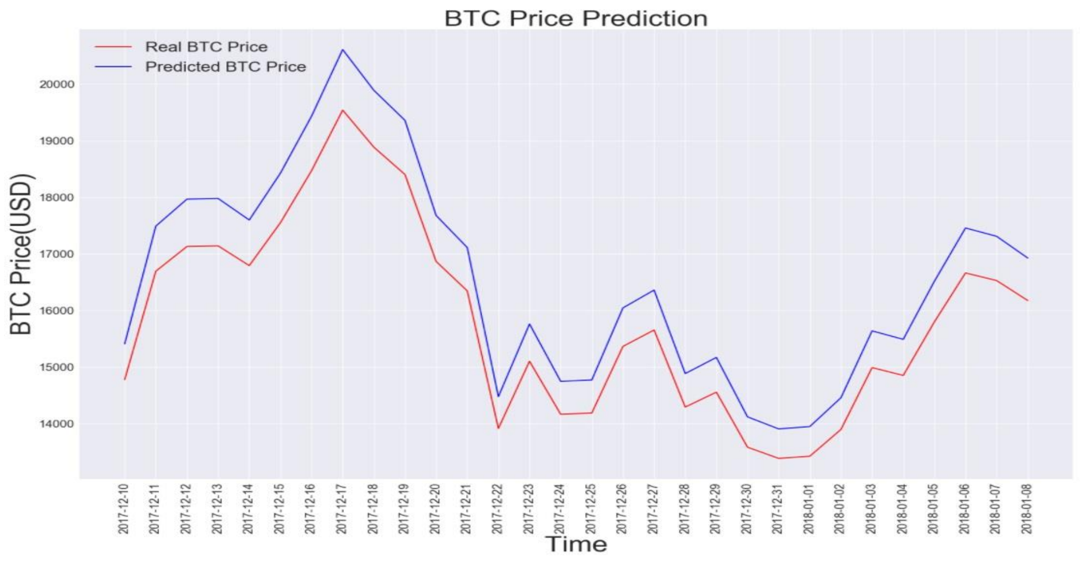
Step 6: Visualization



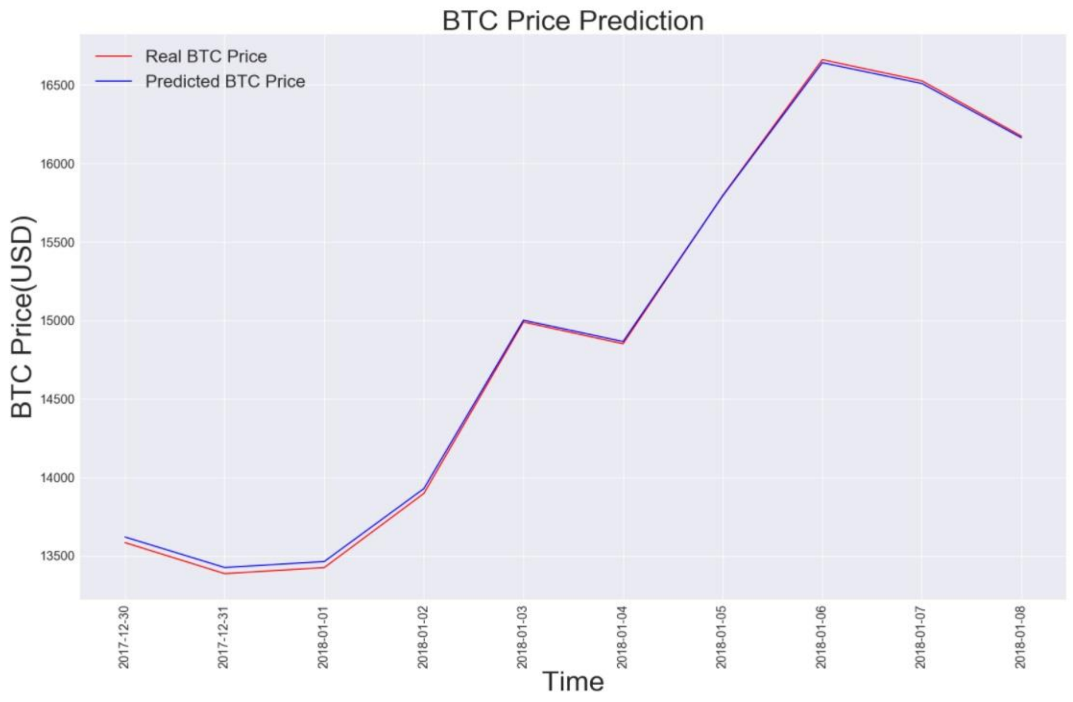
```
# Visualization
plt.figure(figsize=(25,15), dpi=80, facecolor='w', edgecolor='k')
ax = plt.gca()
plt.plot(test_set, color = 'red', label = 'Real BTC Price')
plt.plot(predicted_BTC_price, color = 'blue', label = 'Predicted BTC Price')
plt.title('BTC Price Prediction', fontsize=40)
df_test = df_test.reset_index()
x=df_test.index
labels = df_test['date']
plt.xticks(x, labels, rotation = 'vertical')
for tick in ax.xaxis.get_major_ticks():
    tick.label1.set_fontsize(18)
for tick in ax.yaxis.get_major_ticks():
    tick.label1.set_fontsize(18)
plt.xlabel('Time', fontsize=40)
plt.ylabel('BTC Price(USD)', fontsize=40)
plt.legend(loc=2, prop={'size': 25})
plt.show()
```

5. Results

Prediction days 30:



Prediction days 10:



6. Discussion

The results show that, as the difference between predicted price and actual price is greater when time is further to the training set. This might be happening because the price of Bitcoin is changing exponentially in recent times and the RNN can't keep up with the sudden changes in the price over the long duration. Running RNN multiple time changing the output as model gets over fitted.

7. References

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9

https://www.kaggle.com/mczielinski/bitcoin-historical-data