Deep learning part

Recurrent neural network (RNN) and Long Short-Term Memory(LSTM) for Stock Market Predictions

Colab link: https://colab.research.google.com/drive/11Loo-3jljObW91hqmsUfsPsC9L2DdiQv

The reason why we decided to use the LSTM model is due to the fact that it is capable of predicting outputs based on past states. In our case we are trying to predict future stock values based on past information.

In [0]:

```
# Boost CPU - Remember to set runtime to CPU
!nvidia-smi
```

```
Fri Nov 29 18:55:05 2019
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 440.33.01    Driver Version: 418.67       CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla P100-PCIE...  Off  | 00000000:00:04.0 Off |                    0 |
| N/A   38C    P0    26W / 250W |      0MiB / 16280MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

In [0]:

```
#install gdown
import os.path as path
if not path.exists('M3final.zip'):
 !pip install gdown
 !gdown https://drive.google.com/uc?id=1w4Sf7GXSzVs5Gcci9Dx2GqiT96A0OeeU
 !unzip M3final.zip
else :
 print('data is in places')
```

```
Requirement already satisfied: gdown in /usr/local/lib/python3.6/dist-packages (3.6.4)
Requirement already satisfied: tqdm in /usr/local/lib/python3.6/dist-packages (from gdown
) (4.28.1)
Requirement already satisfied: requests in /usr/local/lib/python3.6/dist-packages (from g
down) (2.21.0)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from gdown)
(1.12.0)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/local/lib/python3.6/dist-pac
kages (from requests->gdown) (3.0.4)
Requirement already satisfied: urllib3<1.25,>=1.21.1 in /usr/local/lib/python3.6/dist-pac
kages (from requests->gdown) (1.24.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-packag
es (from requests->gdown) (2019.9.11)
Requirement already satisfied: idna<2.9,>=2.5 in /usr/local/lib/python3.6/dist-packages (
from requests->gdown) (2.8)
Downloading...
From: https://drive.google.com/uc?id=1w4Sf7GXSzVs5Gcci9Dx2GqiT96A0OeeU
To: /content/M3final.zip
31.8MB [00:00, 87.3MB/s]
Archive:  M3final.zip
  inflating: fundamentals.csv
  inflating: __MACOSX/._fundamentals.csv
```

```
  inflating: prices-split-adjusted.csv
  inflating: __MACOSX/._prices-split-adjusted.csv
  inflating: prices.csv
  inflating: __MACOSX/._prices.csv
  inflating: securities.csv
  inflating: __MACOSX/._securities.csv
```

In [0]:

```
# Import packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
from keras.models import Sequential
from keras.layers import Dense , BatchNormalization , Dropout , Activation
from keras.layers import LSTM , GRU
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from keras.optimizers import Adam , SGD , RMSprop
```

```
Using TensorFlow backend.
```

The default version of TensorFlow in Colab will soon switch to TensorFlow 2.x.
We recommend you upgrade now or ensure your notebook will continue to use TensorFlow 1.x via the `%tensorflow_version 1.x` magic: more info.

In [0]:

```
# Load data
df = pd.read_csv('prices-split-adjusted.csv')
```

In [0]:

```
# Checking the data
df.head()
```

Out[0]:

	date	symbol	open	close	low	high	volume
0	2016-01-05	WLTW	123.430000	125.839996	122.309998	126.250000	2163600.0
1	2016-01-06	WLTW	125.239998	119.980003	119.940002	125.540001	2386400.0
2	2016-01-07	WLTW	116.379997	114.949997	114.930000	119.739998	2489500.0
3	2016-01-08	WLTW	115.480003	116.620003	113.500000	117.440002	2006300.0
4	2016-01-11	WLTW	117.010002	114.970001	114.089996	117.330002	1408600.0

In [0]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 851264 entries, 0 to 851263
Data columns (total 7 columns):
date      851264 non-null object
symbol    851264 non-null object
open      851264 non-null float64
close     851264 non-null float64
low       851264 non-null float64
high      851264 non-null float64
volume    851264 non-null float64
dtypes: float64(5), object(2)
memory usage: 45.5+ MB
```

In [0]:

```
df['date'] = pd.to_datetime(df['date'])
```

In [0]:

```
# Sorting the date in chronological order
df = df.sort_values('date')
```

In [0]:

```
# Again choosing stock of IBM using the ticker/symbol
df = df[df.symbol == 'IBM']
```

In [0]:

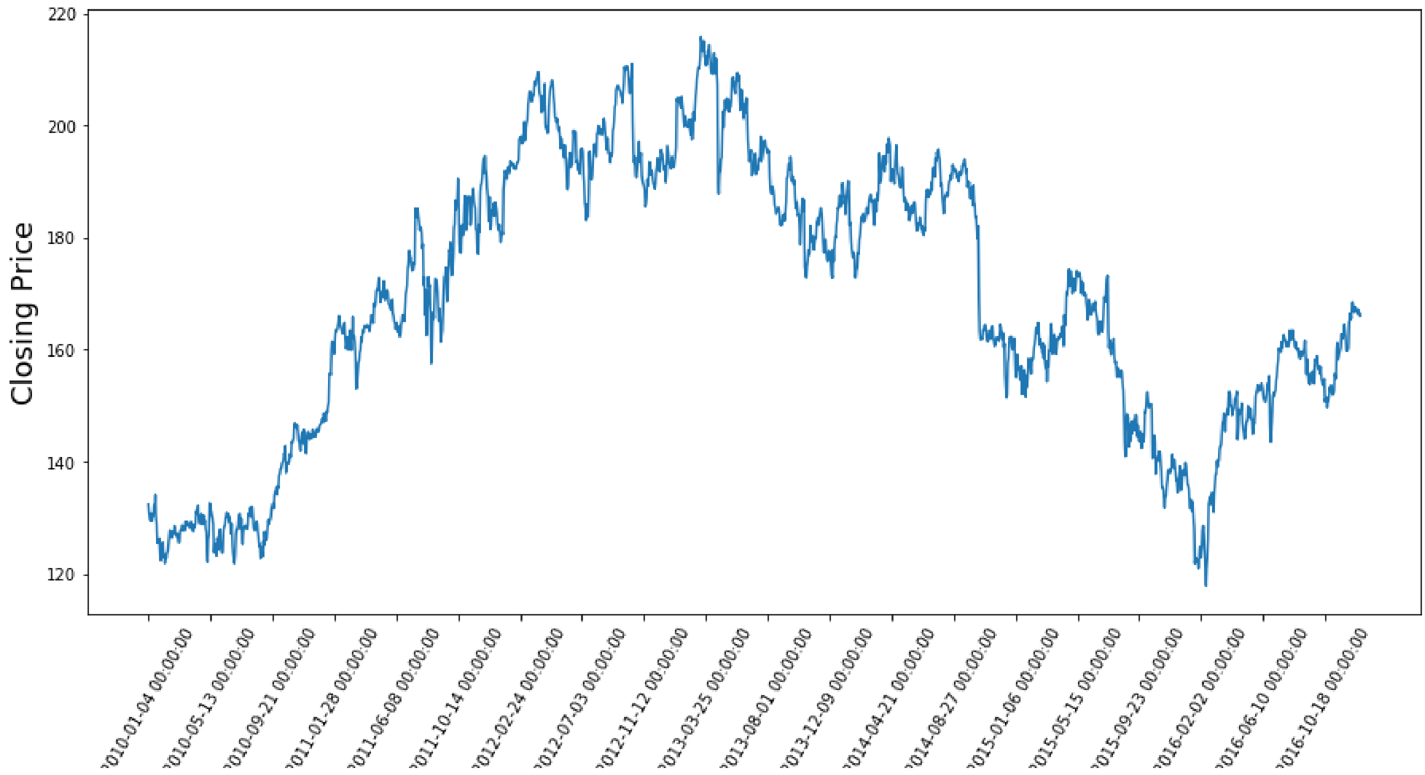
```
# Here we see that there are only IMB stock information
df.head(10)
```

Out[0]:

	date	symbol	open	close	low	high	volume
470	2010-01-04	IBM	131.179993	132.449997	130.850006	132.970001	6155300.0
938	2010-01-05	IBM	131.679993	130.850006	130.100006	131.850006	6841400.0
1406	2010-01-06	IBM	130.679993	130.000000	129.809998	131.490005	5605300.0
1874	2010-01-07	IBM	129.869995	129.550003	128.910004	130.250000	5840600.0
2342	2010-01-08	IBM	129.070007	130.850006	129.050003	130.919998	4197200.0
2810	2010-01-11	IBM	131.059998	129.479996	128.669998	131.059998	5730400.0
3278	2010-01-12	IBM	129.029999	130.509995	129.000000	131.330002	8081500.0
3746	2010-01-13	IBM	130.389999	130.229996	129.160004	131.119995	6455400.0
4214	2010-01-14	IBM	130.550003	132.309998	129.910004	132.710007	7111800.0
4682	2010-01-15	IBM	132.029999	131.779999	131.089996	132.889999	8494400.0

In [0]:

```
# Plot of how the IBM stock price developed over time
plt.figure(figsize = (15,7))
plt.plot(range(df.shape[0]),(df['close']))
plt.xticks(range(0,df.shape[0],90),df['date'].loc[::90],rotation=60) #date shows each 90
days
plt.xlabel('Date',fontsize=18)
plt.ylabel('Closing Price',fontsize=18)
plt.show()
```



In [0]:

```
# Puting the closing stock into an array
closing_stock = np.array(df.iloc[: , 3])
```

In [0]:

```
# Reshape the array
stocks = closing_stock[:]
print(stocks)
stocks = stocks.reshape(len(stocks) , 1)
```

```
[132.449997 130.850006 130.        ... 166.190002 166.600006 165.990005]
```

Spliting data into Training and Test set

We are splitting the data into a 95% training set and 5% test set

In [0]:

```
# Normalizing the data using the min max scaler
# min max scaler sclaes the values of all the data betweet 0 and 1
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))
stocks = scaler.fit_transform(stocks)
```

In [0]:

```
# Splitting the data - 95% train/5% test
train = int(len(stocks) * 0.95)
test = len(stocks) - train
```

In [0]:

```
# Print the train and test size
print(train , test)
```

```
1673 89
```

In [0]:

```
train = stocks[0:train]
print(train)
```

```
[[0.14905562]
 [0.13272085]
 [0.12404289]
 ...
 [0.43032159]
 [0.43297596]
 [0.4206228 ]]
```

In [0]:

```
test = stocks[len(train) : ]
```

In [0]:

```
# Reshaping the array
train = train.reshape(len(train) , 1)
test = test.reshape(len(test) , 1)
```

In [0]:

```
# Checking shape of the train and test set
print(train.shape , test.shape)
```

```
(1673, 1) (89, 1)
```

In [0]:

```
# Defining the function to feed the LSTM with the right input

def process_data(data , n_features):
    X_data, y_data = [], []
    for i in range(len(data)-n_features-1):
        a = data[i:(i+n_features), 0]
        X_data.append(a)
        y_data.append(data[i + n_features, 0])
    return np.array(X_data), np.array(y_data)
```

In [0]:

```
n_features = 2

trainX, trainY = process_data(train, n_features)
testX, testY = process_data(test, n_features)
```

In [0]:

```
print(trainX.shape , trainY.shape , testX.shape , testY.shape)
```

```
(1670, 2) (1670,) (86, 2) (86,)
```

In [0]:

```
# Reshaping train and test set
trainX = trainX.reshape(trainX.shape[0] , 1 ,trainX.shape[1])
testX = testX.reshape(testX.shape[0] , 1 ,testX.shape[1])
```

Making Stock Price Predictions using LSTM

We are making a model to predict the price of IBM stocks using the keras sequential model. The last step is to Initialize our model with the layers we created and define the loss function and gradient descent optimizer. We can also pass other metrics such as "accuracy" that Keras will track for us. Finally, we fit the model using the training and validation we created earlier. The model will be set to 20 epochs which tells the model to loop through our training dataset 20 times, each time optimizing the weights and bias to reduce our loss.

In [0]:

```
# Creating the model
model = Sequential()
model.add(GRU(256 , input_shape = (1 , n_features) , return_sequences=True))
model.add(Dropout(0.2)) #Dropout to prevent gross overfitting - we are excluding 0.2 % of
the data
model.add(LSTM(128)) #Adding the LSTM
model.add(Dropout(0.2))
model.add(Dense(64 ,  activation = 'relu'))
model.add(Dense(32 ,  activation = 'relu'))
model.add(Dense(1))
print(model.summary())
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_b
ackend.py:66: The name tf.get_default_graph is deprecated. Please use tf.compat.v1.get_de
fault_graph instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_b
ackend.py:541: The name tf.placeholder is deprecated. Please use tf.compat.v1.placeholder
instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_b
ackend.py:4432: The name tf.random_uniform is deprecated. Please use tf.random.uniform in
stead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_b
```

```
ackend.py:148: The name tf.placeholder_with_default is deprecated. Please use tf.compat.v
1.placeholder_with_default instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_b
ackend.py:3733: calling dropout (from tensorflow.python.ops.nn_ops) with keep_prob is dep
recated and will be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru_1 (GRU)                  (None, 1, 256)            198912
_________________________________________________________________
dropout_1 (Dropout)          (None, 1, 256)            0
_________________________________________________________________
lstm_1 (LSTM)                (None, 128)               197120
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_2 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 33
=================================================================
Total params: 406,401
Trainable params: 406,401
Non-trainable params: 0
_________________________________________________________________
None
```

The reason why we added two Drop out layers is because Neural Networks have a tendency to overfit

In [0]:

```
# Compiling the model using
model.compile(loss='mean_squared_error', optimizer="Adam" , metrics = ['mean_squared_err
or'])
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/optimizers.py:793: T
he name tf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Optimizer instead
.
```

In [0]:

```
# Fit the model
history = model.fit(trainX, trainY, epochs=20 , batch_size = 64 , validation_data = (tes
tX,testY))
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python/ops
/math_grad.py:1424: where (from tensorflow.python.ops.array_ops) is deprecated and will b
e removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_b
ackend.py:1033: The name tf.assign_add is deprecated. Please use tf.compat.v1.assign_add
instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_b
ackend.py:1020: The name tf.assign is deprecated. Please use tf.compat.v1.assign instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_b
ackend.py:3005: The name tf.Session is deprecated. Please use tf.compat.v1.Session instea
d.

Train on 1670 samples, validate on 86 samples
Epoch 1/20
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_b
ackend.py:190: The name tf.get_default_session is deprecated. Please use tf.compat.v1.get
_default_session instead.
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_b
ackend.py:197: The name tf.ConfigProto is deprecated. Please use tf.compat.v1.ConfigProto
instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_b
ackend.py:207: The name tf.global_variables is deprecated. Please use tf.compat.v1.global
_variables instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_b
ackend.py:216: The name tf.is_variable_initialized is deprecated. Please use tf.compat.v1
.is_variable_initialized instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_b
ackend.py:223: The name tf.variables_initializer is deprecated. Please use tf.compat.v1.v
ariables_initializer instead.

1670/1670 [==============================] - 6s 3ms/step - loss: 0.1296 - mean_squared_er
ror: 0.1296 - val_loss: 0.0057 - val_mean_squared_error: 0.0057
Epoch 2/20
1670/1670 [==============================] - 0s 147us/step - loss: 0.0112 - mean_squared_
error: 0.0112 - val_loss: 0.0015 - val_mean_squared_error: 0.0015
Epoch 3/20
1670/1670 [==============================] - 0s 149us/step - loss: 0.0041 - mean_squared_
error: 0.0041 - val_loss: 2.8943e-04 - val_mean_squared_error: 2.8943e-04
Epoch 4/20
1670/1670 [==============================] - 0s 152us/step - loss: 0.0020 - mean_squared_
error: 0.0020 - val_loss: 3.1915e-04 - val_mean_squared_error: 3.1915e-04
Epoch 5/20
1670/1670 [==============================] - 0s 142us/step - loss: 0.0020 - mean_squared_
error: 0.0020 - val_loss: 3.6848e-04 - val_mean_squared_error: 3.6848e-04
Epoch 6/20
1670/1670 [==============================] - 0s 140us/step - loss: 0.0018 - mean_squared_
error: 0.0018 - val_loss: 3.0035e-04 - val_mean_squared_error: 3.0035e-04
Epoch 7/20
1670/1670 [==============================] - 0s 152us/step - loss: 0.0018 - mean_squared_
error: 0.0018 - val_loss: 2.8796e-04 - val_mean_squared_error: 2.8796e-04
Epoch 8/20
1670/1670 [==============================] - 0s 146us/step - loss: 0.0017 - mean_squared_
error: 0.0017 - val_loss: 3.1229e-04 - val_mean_squared_error: 3.1229e-04
Epoch 9/20
1670/1670 [==============================] - 0s 151us/step - loss: 0.0016 - mean_squared_
error: 0.0016 - val_loss: 2.8644e-04 - val_mean_squared_error: 2.8644e-04
Epoch 10/20
1670/1670 [==============================] - 0s 137us/step - loss: 0.0016 - mean_squared_
error: 0.0016 - val_loss: 3.0014e-04 - val_mean_squared_error: 3.0014e-04
Epoch 11/20
1670/1670 [==============================] - 0s 137us/step - loss: 0.0017 - mean_squared_
error: 0.0017 - val_loss: 3.1931e-04 - val_mean_squared_error: 3.1931e-04
Epoch 12/20
1670/1670 [==============================] - 0s 144us/step - loss: 0.0015 - mean_squared_
error: 0.0015 - val_loss: 3.1147e-04 - val_mean_squared_error: 3.1147e-04
Epoch 13/20
1670/1670 [==============================] - 0s 135us/step - loss: 0.0015 - mean_squared_
error: 0.0015 - val_loss: 3.1643e-04 - val_mean_squared_error: 3.1643e-04
Epoch 14/20
1670/1670 [==============================] - 0s 138us/step - loss: 0.0016 - mean_squared_
error: 0.0016 - val_loss: 3.9829e-04 - val_mean_squared_error: 3.9829e-04
Epoch 15/20
1670/1670 [==============================] - 0s 140us/step - loss: 0.0016 - mean_squared_
error: 0.0016 - val_loss: 3.4852e-04 - val_mean_squared_error: 3.4852e-04
Epoch 16/20
1670/1670 [==============================] - 0s 143us/step - loss: 0.0015 - mean_squared_
error: 0.0015 - val_loss: 3.2216e-04 - val_mean_squared_error: 3.2216e-04
Epoch 17/20
1670/1670 [==============================] - 0s 138us/step - loss: 0.0013 - mean_squared_
error: 0.0013 - val_loss: 2.8272e-04 - val_mean_squared_error: 2.8272e-04
Epoch 18/20
1670/1670 [==============================] - 0s 148us/step - loss: 0.0013 - mean_squared_
error: 0.0013 - val_loss: 3.2557e-04 - val_mean_squared_error: 3.2557e-04
Epoch 19/20
1670/1670 [==============================] - 0s 139us/step - loss: 0.0013 - mean_squared_
```

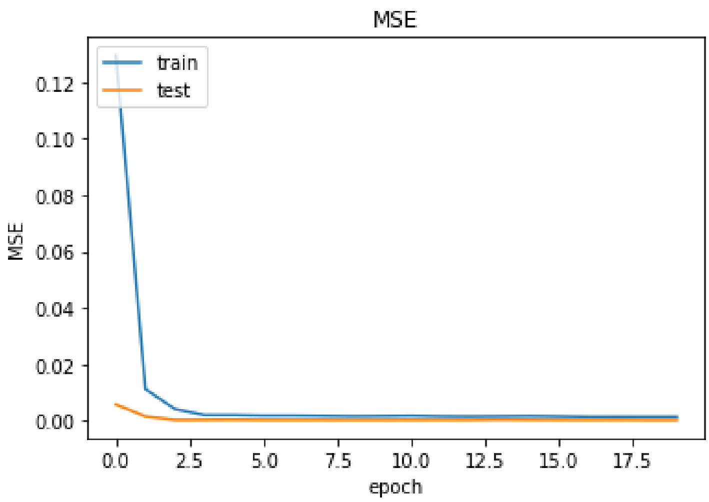
```
error: 0.0013 - val_loss: 2.7481e-04 - val_mean_squared_error: 2.7481e-04
Epoch 20/20
1670/1670 [==============================] - 0s 146us/step - loss: 0.0013 - mean_squared_
error: 0.0013 - val_loss: 2.7623e-04 - val_mean_squared_error: 2.7623e-04
```

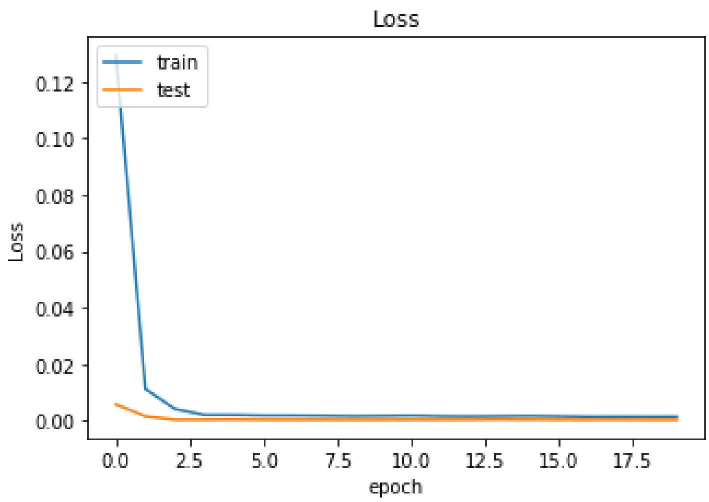
In [0]:

```python
# Plotting the MSE and Loss for the model

# how well the mse is doing for each epoch
plt.plot(history.history['mean_squared_error'])
plt.plot(history.history['val_mean_squared_error'])
plt.title('MSE')
plt.ylabel('MSE')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()


# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Loss')
plt.ylabel('Loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```





In [0]:

```python
# Checking the MSE for the train and test set respectively
def model_score(model, X_train, y_train, X_test, y_test):
    trainScore = model.evaluate(X_train, y_train, verbose=0)
    print('Train Score: %.5f MSE (%.2f RMSE)' % (trainScore[0], math.sqrt(trainScore[0])
))
    testScore = model.evaluate(X_test, y_test, verbose=0)
```

```
    print('Test Score: %.5f MSE (%.2f RMSE)' % (testScore[0], math.sqrt(testScore[0])))
    return trainScore[0], testScore[0]

model_score(model, trainX, trainY , testX, testY)
```

```
Train Score: 0.00060 MSE (0.02 RMSE)
Test Score: 0.00028 MSE (0.02 RMSE)
```

Out[0]:

```
(0.0005982842622995781, 0.00027623464255918597)
```

In [0]:

```
pred = model.predict(testX)
pred = scaler.inverse_transform(pred)
pred[:10]
```

Out[0]:

```
array([[158.52751],
       [159.18913],
       [159.5365 ],
       [159.12733],
       [159.29915],
       [159.55238],
       [160.00127],
       [161.02069],
       [160.03468],
       [157.2072 ]], dtype=float32)
```

In [0]:

```
pred.shape
```

Out[0]:

```
(86, 1)
```

In [0]:

```
testY = testY.reshape(testY.shape[0] , 1)
testY = scaler.inverse_transform(testY)
testY[:10]
```

Out[0]:

```
array([[159.720001],
       [159.399994],
       [158.880005],
       [159.539993],
       [159.550003],
       [160.350006],
       [161.639999],
       [159.      ],
       [155.690002],
       [158.289993]])
```

In [0]:

```
# Using r2 score to see how close the prediction is to the actual price
from sklearn.metrics import r2_score
r2_score(testY, pred)
```

Out[0]:

```
0.8978255813786619
```

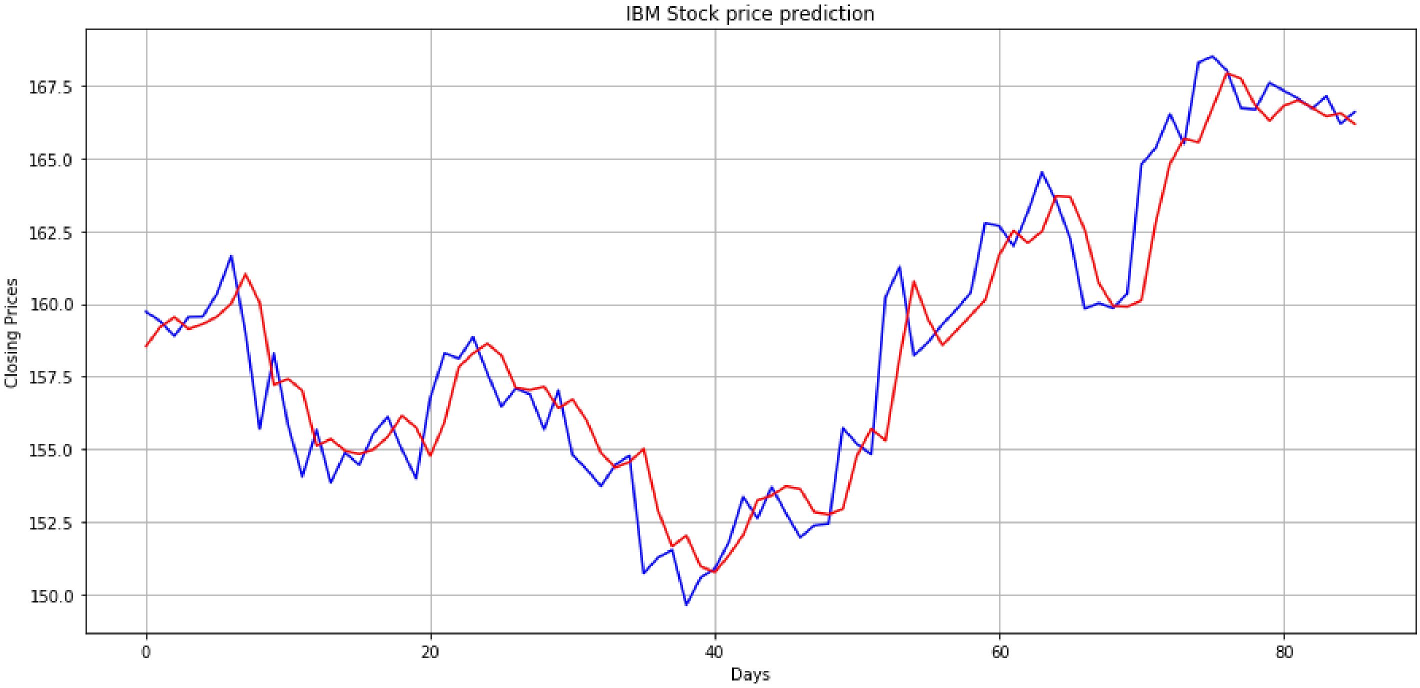
In [0]:

```
# Plotting the predicted price vs actual closing price

plt.rcParams["figure.figsize"] = (15,7)
```

```
plt.plot(testY , 'blue')
plt.plot(pred , 'red')
plt.xlabel('Days')
plt.ylabel('Closing Prices')
plt.title('IBM Stock price prediction')
plt.grid(True)
plt.show()

print("Red - Predicted closing price  ,  Blue - Actual closing price")
```




```
Red - Predicted closing price  ,  Blue - Actual closing price
```

Results

We see that the RNN gives us a MSE of 0.00036, which we are quite satisfied with. We also have the R^2 metrics which comes out to 86.6, something you can see in the plot above is fearly accurate .