Neural Networks & Deep Learning – Final Increment (Project)

Project Title: Stock Market Prediction Using LSTM

Team Members:

1. Kamala Ramesh – 700745451
2. Lnu Rumana Thaskeen- 700742859
3. Srujana Reddy Makutam - 700740914
4. Tejaswini Sankoju – 700726283

Please find below the screenshots of the executions,

In [11]:
```
#Importing the libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import keras
import warnings
warnings.filterwarnings("ignore")
```

In [12]:
```
#Importing the data
train= pd.read_csv('Price_train.csv')
test= pd.read_csv('Price_test.csv')
```

In [13]:
```
train.head()
```

Out[13]:

	Date	Open	High	Low	Close	Volume
0	1/3/2012	325.25	332.83	324.97	663.59	7,380,500
1	1/4/2012	331.27	333.87	329.08	666.45	5,749,400
2	1/5/2012	329.83	330.75	326.89	657.21	6,590,300
3	1/6/2012	328.34	328.77	323.68	648.24	5,405,900
4	1/9/2012	322.04	322.29	309.46	620.76	11,688,800

In [14]:
```
#taking open price from data in 2d array , if we will do train.loc[:, 'open'].values it gives one d array which wont
#be considered in scaling
train_open= train.iloc[:, 1:2].values
```

In [15]:
```
#Scaling the values between 0 to 1
from sklearn.preprocessing import MinMaxScaler
ss= MinMaxScaler(feature_range=(0,1))
train_open_scaled= ss.fit_transform(train_open)
```

In [16]:
```
train_open_scaled[60]
```

Out[16]:
```
array([0.08627874])
```

In [17]:
```
# Feature selection
xtrain=[]
ytrain=[]
for i in range(60,len(train_open_scaled)):
    xtrain.append(train_open_scaled[i-60:i,0])
    ytrain.append(train_open_scaled[i,0])

xtrain, ytrain = np.array(xtrain), np.array(ytrain)
```

In [18]:
```
#Reshaping the train data to make it as input for LTSM layer input_shape(batchzise,timesteps,input_dim)
xtrain= np.reshape(xtrain,(xtrain.shape[0],xtrain.shape[1],1))
```

In [19]:
```
xtrain.shape
```

Out[19]:
```
(1198, 60, 1)
```

Building the LSTM Network

In [20]:
```
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers import Dropout
```

In [21]:
```
#initialisizng the model
regression= Sequential()

#First Input layer and LSTM layer with 0.2% dropout
regression.add(LSTM(units=50,return_sequences=True,kernel_initializer='glorot_uniform',input_shape=(xtrain.shape[1],1)))
regression.add(Dropout(0.2))

# Where:
#     return_sequences: Boolean. Whether to return the last output in the output sequence, or the full sequence.

# Second LSTM layer with 0.2% dropout
regression.add(LSTM(units=50,kernel_initializer='glorot_uniform',return_sequences=True))
regression.add(Dropout(0.2))

#Third LSTM layer with 0.2% dropout
regression.add(LSTM(units=50,kernel_initializer='glorot_uniform',return_sequences=True))
regression.add(Dropout(0.2))

#Fourth LSTM layer with 0.2% dropout, we wont use return sequence true in last layers as we dont want to previous output
regression.add(LSTM(units=50,kernel_initializer='glorot_uniform'))
regression.add(Dropout(0.2))
#Output layer , we wont pass any activation as its continous value model
regression.add(Dense(units=1))

#Compiling the network
regression.compile(optimizer='adam',loss='mean_squared_error')
```

```
#fitting the network
regression.fit(xtrain,ytrain,batch_size=30,epochs=100)
```

```
Epoch 92/100
40/40 [==============================] - 3s 75ms/step - loss: 0.0014
Epoch 93/100
40/40 [==============================] - 3s 73ms/step - loss: 0.0013
Epoch 94/100
40/40 [==============================] - 3s 73ms/step - loss: 0.0016
Epoch 95/100
40/40 [==============================] - 3s 73ms/step - loss: 0.0014
Epoch 96/100
40/40 [==============================] - 3s 79ms/step - loss: 0.0016
Epoch 97/100
40/40 [==============================] - 3s 71ms/step - loss: 0.0013
Epoch 98/100
40/40 [==============================] - 3s 78ms/step - loss: 0.0013
Epoch 99/100
40/40 [==============================] - 3s 71ms/step - loss: 0.0014
Epoch 100/100
40/40 [==============================] - 3s 69ms/step - loss: 0.0014
```

Out[21]:
```
<keras.callbacks.History at 0x1bf11ddca30>
```

In [31]:
```
test_open= test.iloc[:, 1:2].values #taking  open price
total= pd.concat([train['Open'],test['Open']],axis=0) # Concating train and test and then will take last 60 train point
test_input = total[len(total)-len(test)-60:].values
test_input= test_input.reshape(-1,1) # reshaping it to get it transformed
test_input= ss.transform(test_input)
```

In [32]:
```
xtest= []
for i in range(60,80):
    xtest.append(test_input[i-60:i,0]) #creating input for lstm prediction
```

In [33]:
```
xtest= np.array(xtest)
```

In [34]:
```
xtest= np.reshape(xtest,(xtest.shape[0],xtest.shape[1],1))
predicted_value= regression.predict(xtest)
```

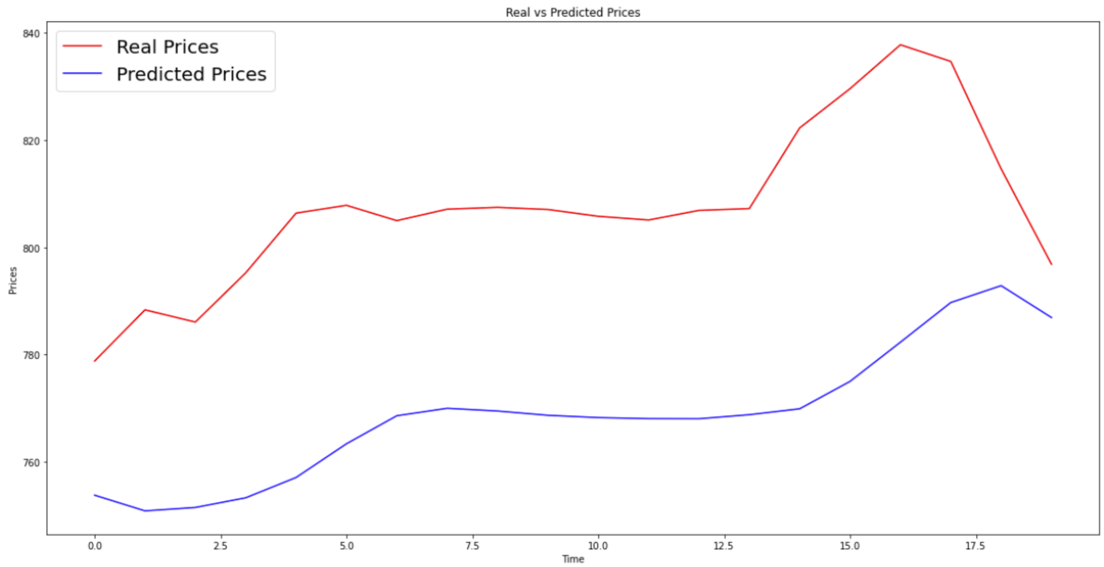
```
1/1 [==============================] - 0s 103ms/step
```

In [35]:
```
predicted_value= ss.inverse_transform(predicted_value)
```

Plotting the data

In [36]:
```
plt.figure(figsize=(20,10))
plt.plot(test_open,'red',label='Real Prices')
plt.plot(predicted_value,'blue',label='Predicted Prices')
plt.xlabel('Time')
plt.ylabel('Prices')
plt.title('Real vs Predicted Prices')
plt.legend(loc='best', fontsize=20)
```

Out[36]: <matplotlib.legend.Legend at 0x1bf1f615cd0>



In [28]:
```
from keras.wrappers.scikit_learn import KerasRegressor
```

In [29]:
```
def reg(optimizer):
    #initialisizng the model
    regression= Sequential()

    #First Input layer and LSTM layer with 0.2% dropout
    regression.add(LSTM(units=50,return_sequences=True,kernel_initializer='glorot_uniform',input_shape=(xtrain.shape[1],1)))
    regression.add(Dropout(0.2))

    # Second LSTM layer with 0.2% dropout
    regression.add(LSTM(units=50,kernel_initializer= 'glorot_uniform',return_sequences=True))
    regression.add(Dropout(0.2))

    #Third LSTM layer with 0.2% dropout
    regression.add(LSTM(units=50,kernel_initializer='glorot_uniform',return_sequences=True))
    regression.add(Dropout(0.2))

    #Fourth LSTM layer with 0.2% dropout, we wont use return sequence true in last layers as we dont want to previous output
    regression.add(LSTM(units=50,kernel_initializer='glorot_uniform'))
    regression.add(Dropout(0.2))
    #Output layer , we wont pass any activation as its continous value model
    regression.add(Dense(units=1))

    #Compiling the network
    regression.compile(optimizer=optimizer,loss='mean_squared_error')

    return regression

model= KerasRegressor(build_fn=reg)
```

In [30]:

```python
from sklearn.model_selection import RandomizedSearchCV
parameters = {'batch_size': [50, 32],
              'epochs': [50, 25],
              'optimizer': ['adam', 'rmsprop','sgd','adadelta']}
grid_search = RandomizedSearchCV(estimator = model,param_distributions=parameters,n_iter=5)
# fitting the model and Calculating the best parameters.
grid_search = grid_search.fit(xtrain, ytrain)
best_parameters = grid_search.best_params_
```

```
38/38 [==============================] - 3s 74ms/step - loss: 0.0038
Epoch 17/25
38/38 [==============================] - 3s 74ms/step - loss: 0.0036
Epoch 18/25
38/38 [==============================] - 3s 78ms/step - loss: 0.0037
Epoch 19/25
38/38 [==============================] - 3s 76ms/step - loss: 0.0034
Epoch 20/25
38/38 [==============================] - 3s 80ms/step - loss: 0.0031
Epoch 21/25
38/38 [==============================] - 3s 77ms/step - loss: 0.0039
Epoch 22/25
38/38 [==============================] - 3s 78ms/step - loss: 0.0032
Epoch 23/25
38/38 [==============================] - 3s 76ms/step - loss: 0.0034
Epoch 24/25
38/38 [==============================] - 3s 74ms/step - loss: 0.0034
Epoch 25/25
38/38 [==============================] - 3s 74ms/step - loss: 0.0030
```

In [35]:

```python
model=grid_search.best_estimator_.fit(xtrain,ytrain)
```

```
Epoch 1/25
1198/1198 [==============================] - 17s 14ms/step - loss: 0.0461
Epoch 2/25
1198/1198 [==============================] - 6s 5ms/step - loss: 0.0152
Epoch 3/25
1198/1198 [==============================] - 6s 5ms/step - loss: 0.0135
Epoch 4/25
1198/1198 [==============================] - 6s 5ms/step - loss: 0.0131
Epoch 5/25
1198/1198 [==============================] - 6s 5ms/step - loss: 0.0105
Epoch 6/25
1198/1198 [==============================] - 7s 6ms/step - loss: 0.0103
Epoch 7/25
1198/1198 [==============================] - 6s 5ms/step - loss: 0.0089
Epoch 8/25
1198/1198 [==============================] - 6s 5ms/step - loss: 0.0091
Epoch 9/25
1198/1198 [==============================] - 6s 5ms/step - loss: 0.0083
Epoch 10/25
1198/1198 [==============================] - 5s 5ms/step - loss: 0.0073
Epoch 11/25
1198/1198 [==============================] - 6s 5ms/step - loss: 0.0078
Epoch 12/25
1198/1198 [==============================] - 5s 4ms/step - loss: 0.0080
Epoch 13/25
1198/1198 [==============================] - 5s 4ms/step - loss: 0.0069
Epoch 14/25
1198/1198 [==============================] - 5s 4ms/step - loss: 0.0070
Epoch 15/25
1198/1198 [==============================] - 5s 4ms/step - loss: 0.0069
Epoch 16/25
1198/1198 [==============================] - 5s 4ms/step - loss: 0.0063
Epoch 17/25
1198/1198 [==============================] - 5s 5ms/step - loss: 0.0063
Epoch 18/25
1198/1198 [==============================] - 5s 5ms/step - loss: 0.0059
Epoch 19/25
1198/1198 [==============================] - 5s 5ms/step - loss: 0.0060
Epoch 20/25
1198/1198 [==============================] - 5s 4ms/step - loss: 0.0057
Epoch 21/25
1198/1198 [==============================] - 5s 4ms/step - loss: 0.0046
Epoch 22/25
1198/1198 [==============================] - 5s 5ms/step - loss: 0.0057
Epoch 23/25
1198/1198 [==============================] - 5s 5ms/step - loss: 0.0046
Epoch 24/25
1198/1198 [==============================] - 5s 4ms/step - loss: 0.0047
Epoch 25/25
1198/1198 [==============================] - 5s 4ms/step - loss: 0.0042
```

In [37]:
```
model
```
Out[37]: <keras.callbacks.History at 0x1a6a6b0fd0>

In [40]:
```
predicted_value= grid_search.predict(xtest)
predicted_value= ss.inverse_transform(predicted_value.reshape(-1,1))


plt.figure(figsize=(20,10))
plt.plot(test_open,'red',label='Real Prices')
plt.plot(predicted_value,'blue',label='Predicted Prices')
plt.xlabel('Time')
plt.ylabel('Prices')
plt.title('Real vs Predicted Prices')
plt.legend(loc='best', fontsize=20)
```
Out[40]: <matplotlib.legend.Legend at 0x1a6d63a358>

Out[40]: <matplotlib.legend.Legend at 0x1a6d63a358>