Problem 2

July 19, 2021

1 RNN for 'Open' price timeseries prediction

- RNN network is recurrent neural network, which works as , it saves the output of particular layer and send it to input or features in order to predict the target/output. Here, it fits the condition of our data, as in our data the the output of 1 day next is calculated based on the data of past three days.
- RNN models are best for time series predction.
- Here, we will use Long Short-Term Memory network , also known as LSTM , for the future ' Open' price prediction. LSTM has the capability of learning the dependencies of long time of previous fed data by remembering them . They have chain like structure and instead of using single layer neural network, it uses combination.
- In our model, LSTM fulfil all the requirements , to predict the next day 'Open' Price. Thus, we will use it .
- With LSTM , we can even predict for more days as it has great capacity in remembering.
- RNN is a solution to feed forward neural networks.
- Data descrition: the data consists of six columns namely 'Open', 'Close' , 'High', 'Low', 'Volume', 'Date' and 1259 rows. Below are some graphs which will show columns nature with respect to time. Here, aim is to predict the next day 'Open' price based on previous three days data. Also, in our case we will use only fours features discardig 'Close' column.

[497]:
```
# importing neccessary liberaries
import numpy as np
import pandas as pd
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Dropout, LSTM, BatchNormalization
from tensorflow.keras.callbacks import TensorBoard, ModelCheckpoint
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
%matplotlib inline
```

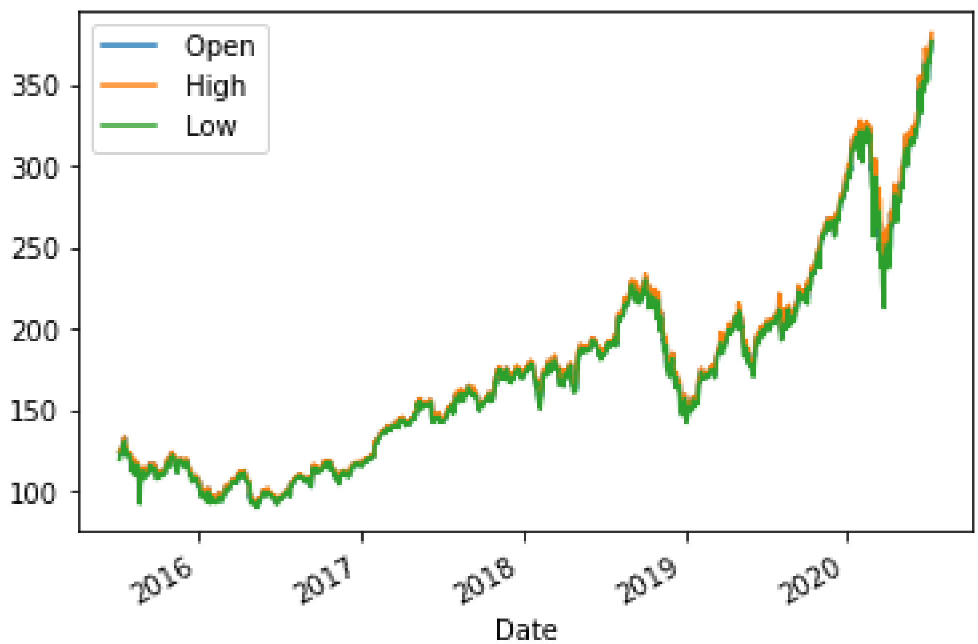
1.0.1 Reading the data:

[498]:
```
data= pd.read_csv('data/q2_dataset.csv')
df= data.copy()
```

1.0.2 Plotting of features:

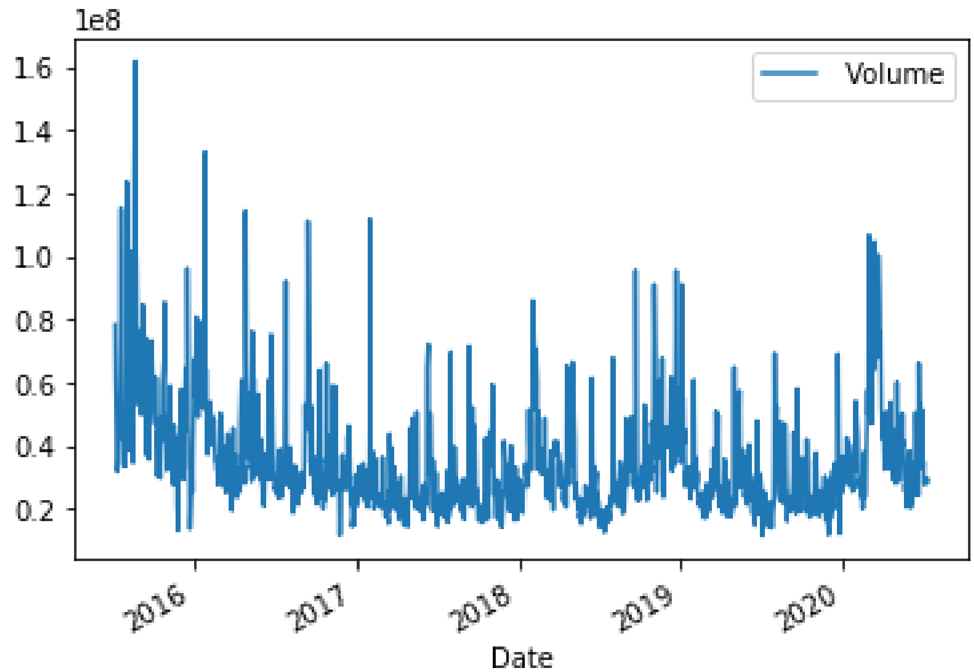
[499]:
```
df1 =  data.copy()
df1['Date']= pd.to_datetime(df1['Date'])
df1.set_index("Date",inplace=True)
df_plot1=df1.iloc[:,2:].tail(1300)
df_plot1.plot.line()
```

[499]: <AxesSubplot:xlabel='Date'>



[500]:
```
df_plot2=df1.iloc[:,1].head(1300)
df_plot2.plot.line()
plt.legend()
```

[500]: <matplotlib.legend.Legend at 0x3d8ce68460>



1.1 Creating dataset with 34 *features(past 3 days* 4 data columns):

Here, we are given six features, namely, ‘Date’,‘Close’,‘Open’,‘Volume’,‘High’,‘Low’. Among these features , the aim is to predict the ‘Open’ price of one day in future using the data of past three days.

Thus, to create a particular dataset, following steps are require, - In order to do so, the two lists will be generated and then they will be iterated through the range starting from the past days ,till the length of days , one less future day to be predicted . As, our data is in ascending order .Thus 1 will be added else, it would be subtracted.

- In the loop, the first generated list ‘x’ will append all the values having last three days data and features , which would make it 3-D numpy array.
- Another list ‘y’ will contain all the one day in future ‘Open’ Price values of ‘x’ list feature and date.
- this function , thus, will return two arrays respresenting feature and target for our model.

This, is how the dataset will be created.

[501]:
```
# Future day whose price is to be predicted
future_day=1
# Count of past days on basis of which target is to be predicted
past_days=3
#Function that returns the feature and target arrays for RNN model
def create_data(df,past_days,future_day):
```

```
    x, y =[],[]
    for i in range(past_days, len(df)-future_day+1):
        x.append(df.iloc[i-past_days:i,0:df.shape[1]])
        y.append(df.iloc[i+future_day-1:i+future_day,3])
    return np.array(x),np.array(y)
x_data , y_data = create_data(df,past_days,future_day)
```

1.1.1 Splitting data:

[502]:
```
# Randomization and Train_test_split of data
x_train , x_test , y_train ,y_test =train_test_split(x_data,y_data,test_size=0.
↪3, random_state=42)
```

1.2 Preprocessing the data:

- During preprocessing of this data , we need four features of past three days and those four features include 'Open','Volume','Low' and 'High'.
- Thus, we will keep these features and remove others . Although the 'Date' column here acts as the index .So, either it should be set as index or remocved.
- Then, to be sure , we converted all the values as float type , suitable to fit in model.
- Then the most important part 'Standard Scaler' is used to scale the values in particular range . else , it would give geat loss and least accuracy while training the network.
- For x_train or x_test data, preprocess function is made , whereas , for y_train and y_test data scale function is made.
- in features preprocessing , as the numpy array is 3-D , thus reshaping is done during scaling the values.
- However, for target , only scaling of 2-D is required.

[503]:
```
# Function that returns Preprocessed Data
def preprocess(df):
    #Close column is not required, so, dropping it
    df= np.delete(df,1,axis=2)

    # Also, date column is here acts as index and not required among 3*4␣
↪features ,so, dropping it
    df= np.delete(df,0,axis=2)

    # Converting all remaining feature values to float
    df=df.astype(float)

    # Standard scaler for 3-D data
    scale= StandardScaler()
    scale_df= scale.fit_transform(df.reshape(-1,df.shape[-1])).reshape(df.shape)
    return scale_df

# Function that return scaled data for 2-D
def scale(df):
```

```
    scale= StandardScaler()
    df= scale.fit_transform(df)
    return df
```

1.2.1 Preprocessing training data:

[504]:
```
# Getting preprocessed data for x_train
x_train = preprocess(x_train)
# Getting scaled data for y_train
y_train = scale(y_train)
```

1.3 Building model:

1.4 All Design steps:

To design following steps are required; - First the model is generated through sequential - Then An LsTM layer is added with number of units as 64 , activation function as 'relu' and input shape as 3*4(features) - After this, another layer of LSTM is added with 32 no. of units and same activation function. But, this time it doies not return any sequences. - then a dropout layer is added, also known as Dropout regularization to drop some elements of data for its better fitting and prediction.Here, 0.2 provided better results among all. - Atlast the Dense output layer is added with 1 unit or to provide same dimension as the target. - then the model is compiled and its loss is calculated. here, we are using mean squared error to calculate loss with 'adam' optimizer. Also, as this is regression model , thus we cannot calculate accyracy through compilation , unlike loss. - Atlast , model summary provided a sneak peek into how the layers will behave inside with their dimensions.

So, this is how our optimal model is designed.

[529]:
```
# Building model with 4 layers
model=Sequential()
model.add(LSTM(64,activation='relu',input_shape=(x_train.shape[1],x_train.
 ↪shape[2]),return_sequences=True))
model.add(LSTM(32,activation='relu',return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(y_train.shape[1]))
model.compile(loss='mse',optimizer='adam')
model.summary()
```

```
Model: "sequential_47"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_88 (LSTM)               (None, 3, 64)             17664

_________________________________________________________________
lstm_89 (LSTM)               (None, 32)                12416

_________________________________________________________________
dropout_46 (Dropout)         (None, 32)                0

_________________________________________________________________
```

```
dense_43 (Dense)             (None, 1)                 33
=================================================================
Total params: 30,113
Trainable params: 30,113
Non-trainable params: 0
_________________________________________________________________
```

1.4.1 Training model:

1.5 Architecture of network:

- In this LSTM model we used , the preprocessed and splitted data with feature and target arrays.
- As, per the epochs , 10 can be used to show that where the model converges. the less the loss, the more accurate will be the predictions. Although, adding more number of epochs will provide even less loss, but , then it will become the case of overfitting. So, 100 epochs are used, with 3 iterations.
- Then, the batch_size is used as 32 , because using 64 or above converge a little late and on more number of epochs. Although, 32 is said to be good batch size for training small models.
- Verbose here is used as 2. it didnt made much difference vem if set to 0 and 1. They are the called as the total count of steps before prediction round finished declaration.

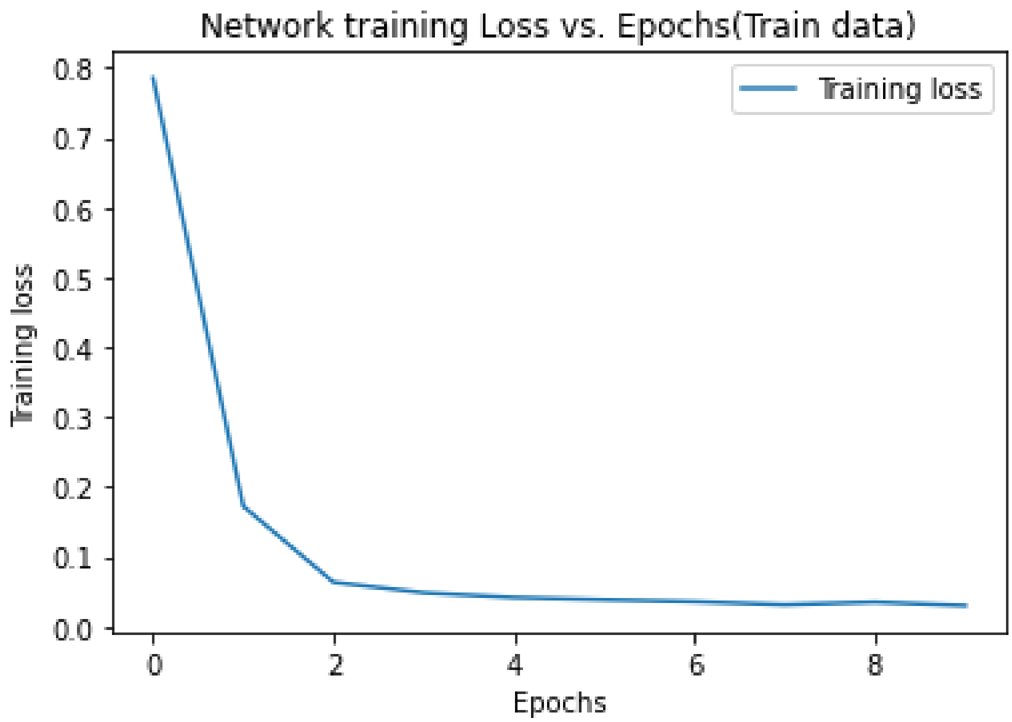
[530]:

```python
# Returns the loss of training data with every epoch
history=model.fit(x_train,y_train,epochs=10,batch_size=32,verbose=2)
```

```
Epoch 1/10
28/28 - 145s - loss: 0.7853
Epoch 2/10
28/28 - 1s - loss: 0.1724
Epoch 3/10
28/28 - 1s - loss: 0.0636
Epoch 4/10
28/28 - 0s - loss: 0.0485
Epoch 5/10
28/28 - 0s - loss: 0.0416
Epoch 6/10
28/28 - 0s - loss: 0.0384
Epoch 7/10
28/28 - 0s - loss: 0.0357
Epoch 8/10
28/28 - 0s - loss: 0.0313
Epoch 9/10
28/28 - 0s - loss: 0.0350
Epoch 10/10
28/28 - 0s - loss: 0.0300
```

1.5.1 Plotting the training data model loss:

[531]:
```
plt.plot(history.history['loss'],label='Training loss')
plt.xlabel('Epochs')
plt.ylabel('Training loss')
plt.title('Network training Loss vs. Epochs(Train data)')
plt.legend()
plt.show()
```



Final loss :

[517]:
```
history=model.fit(x_train,y_train,epochs=100,batch_size=32,verbose=2)
```

```
Epoch 1/100
28/28 - 0s - loss: 0.0180
Epoch 2/100
28/28 - 0s - loss: 0.0174
Epoch 3/100
28/28 - 0s - loss: 0.0204
Epoch 4/100
28/28 - 0s - loss: 0.0197
Epoch 5/100
28/28 - 0s - loss: 0.0203
Epoch 6/100
28/28 - 0s - loss: 0.0205
```

```
Epoch 7/100
28/28 - 0s - loss: 0.0220
Epoch 8/100
28/28 - 0s - loss: 0.0203
Epoch 9/100
28/28 - 0s - loss: 0.0231
Epoch 10/100
28/28 - 0s - loss: 0.0194
Epoch 11/100
28/28 - 0s - loss: 0.0202
Epoch 12/100
28/28 - 0s - loss: 0.0191
Epoch 13/100
28/28 - 0s - loss: 0.0207
Epoch 14/100
28/28 - 0s - loss: 0.0178
Epoch 15/100
28/28 - 0s - loss: 0.0230
Epoch 16/100
28/28 - 0s - loss: 0.0222
Epoch 17/100
28/28 - 0s - loss: 0.0202
Epoch 18/100
28/28 - 0s - loss: 0.0205
Epoch 19/100
28/28 - 0s - loss: 0.0214
Epoch 20/100
28/28 - 0s - loss: 0.0224
Epoch 21/100
28/28 - 0s - loss: 0.0219
Epoch 22/100
28/28 - 0s - loss: 0.0213
Epoch 23/100
28/28 - 0s - loss: 0.0174
Epoch 24/100
28/28 - 0s - loss: 0.0200
Epoch 25/100
28/28 - 0s - loss: 0.0210
Epoch 26/100
28/28 - 0s - loss: 0.0203
Epoch 27/100
28/28 - 0s - loss: 0.0158
Epoch 28/100
28/28 - 0s - loss: 0.0257
Epoch 29/100
28/28 - 0s - loss: 0.0193
Epoch 30/100
28/28 - 0s - loss: 0.0207
```

```
Epoch 31/100
28/28 - 0s - loss: 0.0208
Epoch 32/100
28/28 - 0s - loss: 0.0213
Epoch 33/100
28/28 - 0s - loss: 0.0212
Epoch 34/100
28/28 - 0s - loss: 0.0233
Epoch 35/100
28/28 - 0s - loss: 0.0201
Epoch 36/100
28/28 - 0s - loss: 0.0217
Epoch 37/100
28/28 - 0s - loss: 0.0250
Epoch 38/100
28/28 - 0s - loss: 0.0224
Epoch 39/100
28/28 - 0s - loss: 0.0229
Epoch 40/100
28/28 - 0s - loss: 0.0227
Epoch 41/100
28/28 - 0s - loss: 0.0205
Epoch 42/100
28/28 - 0s - loss: 0.0197
Epoch 43/100
28/28 - 0s - loss: 0.0220
Epoch 44/100
28/28 - 0s - loss: 0.0238
Epoch 45/100
28/28 - 0s - loss: 0.0228
Epoch 46/100
28/28 - 0s - loss: 0.0219
Epoch 47/100
28/28 - 0s - loss: 0.0215
Epoch 48/100
28/28 - 0s - loss: 0.0200
Epoch 49/100
28/28 - 0s - loss: 0.0184
Epoch 50/100
28/28 - 0s - loss: 0.0213
Epoch 51/100
28/28 - 0s - loss: 0.0203
Epoch 52/100
28/28 - 0s - loss: 0.0181
Epoch 53/100
28/28 - 0s - loss: 0.0240
Epoch 54/100
28/28 - 0s - loss: 0.0229
```

```
Epoch 55/100
28/28 - 0s - loss: 0.0199
Epoch 56/100
28/28 - 0s - loss: 0.0182
Epoch 57/100
28/28 - 0s - loss: 0.0195
Epoch 58/100
28/28 - 0s - loss: 0.0241
Epoch 59/100
28/28 - 0s - loss: 0.0275
Epoch 60/100
28/28 - 0s - loss: 0.0227
Epoch 61/100
28/28 - 0s - loss: 0.0201
Epoch 62/100
28/28 - 0s - loss: 0.0202
Epoch 63/100
28/28 - 0s - loss: 0.0184
Epoch 64/100
28/28 - 0s - loss: 0.0195
Epoch 65/100
28/28 - 0s - loss: 0.0221
Epoch 66/100
28/28 - 0s - loss: 0.0196
Epoch 67/100
28/28 - 0s - loss: 0.0242
Epoch 68/100
28/28 - 0s - loss: 0.0209
Epoch 69/100
28/28 - 0s - loss: 0.0214
Epoch 70/100
28/28 - 0s - loss: 0.0217
Epoch 71/100
28/28 - 0s - loss: 0.0240
Epoch 72/100
28/28 - 0s - loss: 0.0192
Epoch 73/100
28/28 - 0s - loss: 0.0170
Epoch 74/100
28/28 - 0s - loss: 0.0206
Epoch 75/100
28/28 - 0s - loss: 0.0211
Epoch 76/100
28/28 - 0s - loss: 0.0208
Epoch 77/100
28/28 - 0s - loss: 0.0218
Epoch 78/100
28/28 - 0s - loss: 0.0196
```

```
Epoch 79/100
28/28 - 0s - loss: 0.0191
Epoch 80/100
28/28 - 0s - loss: 0.0234
Epoch 81/100
28/28 - 0s - loss: 0.0200
Epoch 82/100
28/28 - 0s - loss: 0.0184
Epoch 83/100
28/28 - 0s - loss: 0.0213
Epoch 84/100
28/28 - 0s - loss: 0.0200
Epoch 85/100
28/28 - 0s - loss: 0.0217
Epoch 86/100
28/28 - 0s - loss: 0.0217
Epoch 87/100
28/28 - 0s - loss: 0.0194
Epoch 88/100
28/28 - 0s - loss: 0.0186
Epoch 89/100
28/28 - 0s - loss: 0.0219
Epoch 90/100
28/28 - 0s - loss: 0.0196
Epoch 91/100
28/28 - 0s - loss: 0.0211
Epoch 92/100
28/28 - 0s - loss: 0.0198
Epoch 93/100
28/28 - 0s - loss: 0.0229
Epoch 94/100
28/28 - 0s - loss: 0.0238
Epoch 95/100
28/28 - 0s - loss: 0.0195
Epoch 96/100
28/28 - 0s - loss: 0.0172
Epoch 97/100
28/28 - 0s - loss: 0.0182
Epoch 98/100
28/28 - 0s - loss: 0.0251
Epoch 99/100
28/28 - 0s - loss: 0.0238
Epoch 100/100
28/28 - 0s - loss: 0.0201
```

[518]:
```
print('The final loss of network is: ',history.history['loss'][-1])
```

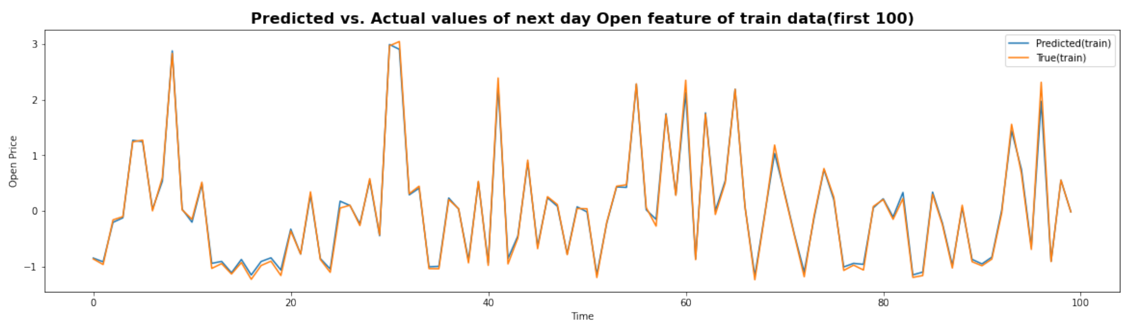
```
The final loss of network is:  0.020050106570124626
```

1.6 Output of training loop:

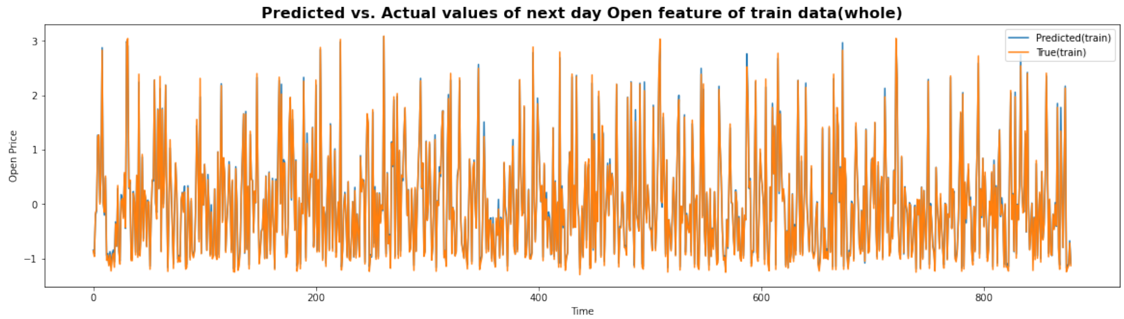
Here, the loss decreases significantly , from 0.6 to 0.0200 .After , 10 epochs we were able to get the minimum loss of 0.02 and after three iterations with 100 epochs we, get the last loss as 0.0200. Thus, 0.02 is a good loss , and it may even converge furher but that will lead to the case of overfiting the data. So, at around 10 epochs we reached the loss of 0.02. Therefore, our final output without overfitting data anymore is 0.02 value of loss. Also, from below plots it is visible that the data predicted is almost as same as the True data. The difference is more visible through first plot of 100 values only.

1.6.1 Predicting and comparing the training values:

[519]:
```
pred_train = model.predict(x_train)
plt.figure(figsize=(20,5))
plt.plot(pred_train[:100], label= 'Predicted(train)')
plt.plot(y_train[:100],label='True(train)')
plt.xlabel('Time')
plt.ylabel('Open Price')
plt.legend()
plt.title('Predicted vs. Actual values of next day Open feature of train␣
 ↪data(first 100)', fontsize=16, fontweight='bold')
plt.show()
```



[520]:
```
plt.figure(figsize=(20,5))
plt.plot(pred_train, label= 'Predicted(train)')
plt.plot(y_train,label='True(train)')
plt.xlabel('Time')
plt.ylabel('Open Price')
plt.legend()
plt.title('Predicted vs. Actual values of next day Open feature of train␣
 ↪data(whole)', fontsize=16, fontweight='bold')
plt.show()
```



1.6.2 Test Data:

[521]:
```
## preprocessing the test data
x_test = preprocess(x_test)
y_test = scale(y_test)

## Train the test data network
history_test=model.fit(x_test,y_test,epochs=100,batch_size=32,verbose=2)
```

```
Epoch 1/100
12/12 - 0s - loss: 0.0213
Epoch 2/100
12/12 - 0s - loss: 0.0216
Epoch 3/100
12/12 - 0s - loss: 0.0187
Epoch 4/100
12/12 - 0s - loss: 0.0210
Epoch 5/100
12/12 - 0s - loss: 0.0204
Epoch 6/100
12/12 - 0s - loss: 0.0238
Epoch 7/100
12/12 - 0s - loss: 0.0184
Epoch 8/100
12/12 - 0s - loss: 0.0171
Epoch 9/100
12/12 - 0s - loss: 0.0213
Epoch 10/100
12/12 - 0s - loss: 0.0200
Epoch 11/100
12/12 - 0s - loss: 0.0202
Epoch 12/100
12/12 - 0s - loss: 0.0250
Epoch 13/100
12/12 - 0s - loss: 0.0216
Epoch 14/100
```

```
12/12 - 0s - loss: 0.0229
Epoch 15/100
12/12 - 0s - loss: 0.0210
Epoch 16/100
12/12 - 0s - loss: 0.0213
Epoch 17/100
12/12 - 0s - loss: 0.0183
Epoch 18/100
12/12 - 0s - loss: 0.0204
Epoch 19/100
12/12 - 0s - loss: 0.0219
Epoch 20/100
12/12 - 0s - loss: 0.0281
Epoch 21/100
12/12 - 0s - loss: 0.0221
Epoch 22/100
12/12 - 0s - loss: 0.0182
Epoch 23/100
12/12 - 0s - loss: 0.0207
Epoch 24/100
12/12 - 0s - loss: 0.0182
Epoch 25/100
12/12 - 0s - loss: 0.0195
Epoch 26/100
12/12 - 0s - loss: 0.0262
Epoch 27/100
12/12 - 0s - loss: 0.0231
Epoch 28/100
12/12 - 0s - loss: 0.0179
Epoch 29/100
12/12 - 0s - loss: 0.0226
Epoch 30/100
12/12 - 0s - loss: 0.0197
Epoch 31/100
12/12 - 0s - loss: 0.0245
Epoch 32/100
12/12 - 0s - loss: 0.0200
Epoch 33/100
12/12 - 0s - loss: 0.0210
Epoch 34/100
12/12 - 0s - loss: 0.0190
Epoch 35/100
12/12 - 0s - loss: 0.0169
Epoch 36/100
12/12 - 0s - loss: 0.0222
Epoch 37/100
12/12 - 0s - loss: 0.0222
Epoch 38/100
```

```
12/12 - 0s - loss: 0.0183
Epoch 39/100
12/12 - 0s - loss: 0.0192
Epoch 40/100
12/12 - 0s - loss: 0.0242
Epoch 41/100
12/12 - 0s - loss: 0.0227
Epoch 42/100
12/12 - 0s - loss: 0.0188
Epoch 43/100
12/12 - 0s - loss: 0.0179
Epoch 44/100
12/12 - 0s - loss: 0.0239
Epoch 45/100
12/12 - 0s - loss: 0.0210
Epoch 46/100
12/12 - 0s - loss: 0.0216
Epoch 47/100
12/12 - 0s - loss: 0.0242
Epoch 48/100
12/12 - 0s - loss: 0.0188
Epoch 49/100
12/12 - 0s - loss: 0.0243
Epoch 50/100
12/12 - 0s - loss: 0.0196
Epoch 51/100
12/12 - 0s - loss: 0.0166
Epoch 52/100
12/12 - 0s - loss: 0.0243
Epoch 53/100
12/12 - 0s - loss: 0.0211
Epoch 54/100
12/12 - 0s - loss: 0.0235
Epoch 55/100
12/12 - 0s - loss: 0.0211
Epoch 56/100
12/12 - 0s - loss: 0.0226
Epoch 57/100
12/12 - 0s - loss: 0.0173
Epoch 58/100
12/12 - 0s - loss: 0.0205
Epoch 59/100
12/12 - 0s - loss: 0.0210
Epoch 60/100
12/12 - 0s - loss: 0.0207
Epoch 61/100
12/12 - 0s - loss: 0.0230
Epoch 62/100
```

```
12/12 - 0s - loss: 0.0196
Epoch 63/100
12/12 - 0s - loss: 0.0221
Epoch 64/100
12/12 - 0s - loss: 0.0233
Epoch 65/100
12/12 - 0s - loss: 0.0186
Epoch 66/100
12/12 - 0s - loss: 0.0195
Epoch 67/100
12/12 - 0s - loss: 0.0213
Epoch 68/100
12/12 - 0s - loss: 0.0227
Epoch 69/100
12/12 - 0s - loss: 0.0231
Epoch 70/100
12/12 - 0s - loss: 0.0239
Epoch 71/100
12/12 - 0s - loss: 0.0214
Epoch 72/100
12/12 - 0s - loss: 0.0200
Epoch 73/100
12/12 - 0s - loss: 0.0232
Epoch 74/100
12/12 - 0s - loss: 0.0188
Epoch 75/100
12/12 - 0s - loss: 0.0199
Epoch 76/100
12/12 - 0s - loss: 0.0165
Epoch 77/100
12/12 - 0s - loss: 0.0202
Epoch 78/100
12/12 - 0s - loss: 0.0231
Epoch 79/100
12/12 - 0s - loss: 0.0222
Epoch 80/100
12/12 - 0s - loss: 0.0178
Epoch 81/100
12/12 - 0s - loss: 0.0210
Epoch 82/100
12/12 - 0s - loss: 0.0219
Epoch 83/100
12/12 - 0s - loss: 0.0209
Epoch 84/100
12/12 - 0s - loss: 0.0205
Epoch 85/100
12/12 - 0s - loss: 0.0225
Epoch 86/100
```

```
12/12 - 0s - loss: 0.0279
Epoch 87/100
12/12 - 0s - loss: 0.0199
Epoch 88/100
12/12 - 0s - loss: 0.0249
Epoch 89/100
12/12 - 0s - loss: 0.0230
Epoch 90/100
12/12 - 0s - loss: 0.0225
Epoch 91/100
12/12 - 0s - loss: 0.0223
Epoch 92/100
12/12 - 0s - loss: 0.0210
Epoch 93/100
12/12 - 0s - loss: 0.0218
Epoch 94/100
12/12 - 0s - loss: 0.0261
Epoch 95/100
12/12 - 0s - loss: 0.0230
Epoch 96/100
12/12 - 0s - loss: 0.0208
Epoch 97/100
12/12 - 0s - loss: 0.0225
Epoch 98/100
12/12 - 0s - loss: 0.0247
Epoch 99/100
12/12 - 0s - loss: 0.0144
Epoch 100/100
12/12 - 0s - loss: 0.0172
```

1.6.3 Evaluating the process data:

[523]:
```
test_evaluation = model.evaluate(x_test,y_test,batch_size=32)
print('The loss while evaluation of test data is: ',test_evaluation)
```

```
12/12 [==============================] - 4s 8ms/step - loss: 0.0045
The loss while evaluation of test data is:  0.004515213891863823
```

1.7 Output of test data:

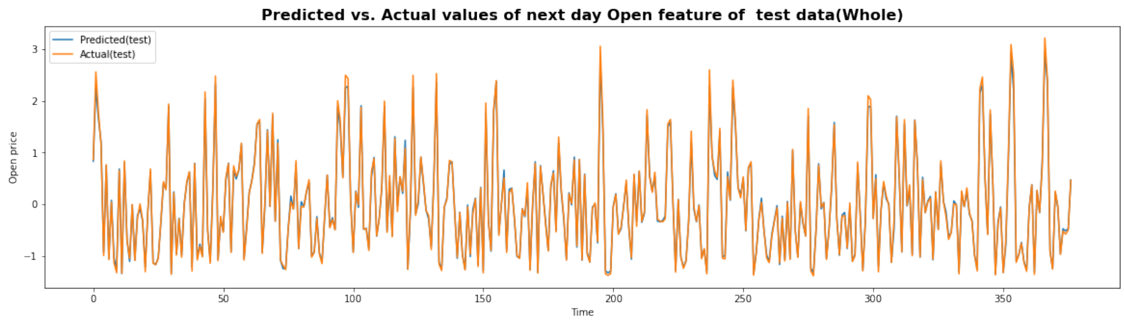
Here, test data covereges till 0.01 that is the even less loss than training data. Thus, it performs slightly better than training one. Also, the below plots are showing the similarilty of data. it is overlapping means , it is predicting correctly. Also, here loss during evaluation is founs as 0.0045. even less than the loss where test model coverges. Therefore, test dataset performs a bit better than train dataset, as visible from the predictions and graph.

1.7.1 Predicting the test values:

[524]:
```
pred_test= model.predict(x_test)
```

1.7.2 Plotting and comapring True vs. Predicted test data target values:

[525]:
```
plt.figure(figsize=(20,5))
plt.plot(pred_test, label= 'Predicted(test)')
plt.plot(y_test,label='Actual(test)')
plt.legend()
plt.xlabel('Time')
plt.ylabel('Open price')
plt.title('Predicted vs. Actual values of next day Open feature of  test␣
 ↪data(Whole)', fontsize=16, fontweight='bold')
plt.show()
```



1.7.3 Using more days as features:

1.7.4 Data preparation:

[526]:
```
# Creating dataset with past days as 30 and future prediction for day as 1
x_data_more , y_data_more = create_data(df,30,1)

# Splitting it into same 70-30 ratio
x_train_more , x_test_more , y_train_more ,y_test_more␣
 ↪=train_test_split(x_data_more,y_data_more,test_size=0.3, random_state=42)

# Preprocessing the features
x_train_more = preprocess(x_data_more)

# scaling the target price value
y_train_more = scale(y_data_more)
```

1.7.5 Model Building for more days:

[533]:
```
model_more=Sequential()
model_more.add(LSTM(64,activation='relu',input_shape=(x_train_more.
 ↪shape[1],x_train_more.shape[2]),return_sequences=True))
# model_more.add(Dropout(0.2))
model_more.add(LSTM(32,activation='relu',return_sequences=False))
model_more.add(Dropout(0.2))

model_more.add(Dense(y_train_more.shape[1]))
model_more.compile(loss='mse',optimizer='adam')
model_more.summary()
```

```
Model: "sequential_48"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_90 (LSTM)               (None, 30, 64)            17664
_________________________________________________________________
lstm_91 (LSTM)               (None, 32)                12416
_________________________________________________________________
dropout_47 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_44 (Dense)             (None, 1)                 33
=================================================================
Total params: 30,113
Trainable params: 30,113
Non-trainable params: 0
_________________________________________________________________
```

1.8 Training model for more days:

In training for more days , it didnt converge as significantely as training for 3 days. Also, it started from 0.2 instead of 0.6 as in the case for three days. Furthermore, its loss values fluctuate between 0.02 and 0.03 even after 100 epochs, whereas, in case of 3 days, it converged after 10 epochs. Thus, in this case the less days model convered earlier or learmed faster as compared to the model with 30 or more days. It is visible from the graph , that even after large epochs it convered near 0.03. It may because of more features , as in case of there are twelwe features for data, but here there are , 30*4 = 120 features for data. So, more the features , less the accuracy in most cases. Apart from this , everything varies from data to data.

[534]:
```
more_days=model_more.
 ↪fit(x_train_more,y_train_more,epochs=100,batch_size=32,verbose=2)
```

```
Epoch 1/100
39/39 - 124s - loss: 0.2898
Epoch 2/100
39/39 - 3s - loss: 0.0582
Epoch 3/100
```

```
39/39 - 2s - loss: 0.0496
Epoch 4/100
39/39 - 1s - loss: 0.0451
Epoch 5/100
39/39 - 1s - loss: 0.0411
Epoch 6/100
39/39 - 1s - loss: 0.0433
Epoch 7/100
39/39 - 1s - loss: 0.0391
Epoch 8/100
39/39 - 1s - loss: 0.0369
Epoch 9/100
39/39 - 1s - loss: 0.0361
Epoch 10/100
39/39 - 1s - loss: 0.0386
Epoch 11/100
39/39 - 1s - loss: 0.0374
Epoch 12/100
39/39 - 1s - loss: 0.0340
Epoch 13/100
39/39 - 1s - loss: 0.0338
Epoch 14/100
39/39 - 1s - loss: 0.0379
Epoch 15/100
39/39 - 1s - loss: 0.0343
Epoch 16/100
39/39 - 1s - loss: 0.0315
Epoch 17/100
39/39 - 1s - loss: 0.0373
Epoch 18/100
39/39 - 1s - loss: 0.0300
Epoch 19/100
39/39 - 1s - loss: 0.0332
Epoch 20/100
39/39 - 1s - loss: 0.0325
Epoch 21/100
39/39 - 1s - loss: 0.0350
Epoch 22/100
39/39 - 1s - loss: 0.0329
Epoch 23/100
39/39 - 1s - loss: 0.0397
Epoch 24/100
39/39 - 1s - loss: 0.0339
Epoch 25/100
39/39 - 1s - loss: 0.0338
Epoch 26/100
39/39 - 1s - loss: 0.0342
Epoch 27/100
```

```
39/39 - 1s - loss: 0.0316
Epoch 28/100
39/39 - 1s - loss: 0.0333
Epoch 29/100
39/39 - 1s - loss: 0.0276
Epoch 30/100
39/39 - 1s - loss: 0.0357
Epoch 31/100
39/39 - 1s - loss: 0.0331
Epoch 32/100
39/39 - 1s - loss: 0.0315
Epoch 33/100
39/39 - 1s - loss: 0.0310
Epoch 34/100
39/39 - 1s - loss: 0.0314
Epoch 35/100
39/39 - 1s - loss: 0.0336
Epoch 36/100
39/39 - 1s - loss: 0.0305
Epoch 37/100
39/39 - 1s - loss: 0.0299
Epoch 38/100
39/39 - 1s - loss: 0.0308
Epoch 39/100
39/39 - 1s - loss: 0.0333
Epoch 40/100
39/39 - 2s - loss: 0.0301
Epoch 41/100
39/39 - 1s - loss: 0.0298
Epoch 42/100
39/39 - 1s - loss: 0.0301
Epoch 43/100
39/39 - 1s - loss: 0.0324
Epoch 44/100
39/39 - 1s - loss: 0.0294
Epoch 45/100
39/39 - 1s - loss: 0.0321
Epoch 46/100
39/39 - 1s - loss: 0.0343
Epoch 47/100
39/39 - 1s - loss: 0.0323
Epoch 48/100
39/39 - 1s - loss: 0.0350
Epoch 49/100
39/39 - 1s - loss: 0.0319
Epoch 50/100
39/39 - 1s - loss: 0.0298
Epoch 51/100
```

```
39/39 - 1s - loss: 0.0287
Epoch 52/100
39/39 - 1s - loss: 0.0319
Epoch 53/100
39/39 - 1s - loss: 0.0309
Epoch 54/100
39/39 - 1s - loss: 0.0302
Epoch 55/100
39/39 - 1s - loss: 0.0326
Epoch 56/100
39/39 - 1s - loss: 0.0283
Epoch 57/100
39/39 - 1s - loss: 0.0301
Epoch 58/100
39/39 - 1s - loss: 0.0385
Epoch 59/100
39/39 - 1s - loss: 0.0322
Epoch 60/100
39/39 - 1s - loss: 0.0275
Epoch 61/100
39/39 - 1s - loss: 0.0358
Epoch 62/100
39/39 - 1s - loss: 0.0329
Epoch 63/100
39/39 - 1s - loss: 0.0292
Epoch 64/100
39/39 - 1s - loss: 0.0308
Epoch 65/100
39/39 - 1s - loss: 0.0305
Epoch 66/100
39/39 - 1s - loss: 0.0305
Epoch 67/100
39/39 - 1s - loss: 0.0296
Epoch 68/100
39/39 - 1s - loss: 0.0321
Epoch 69/100
39/39 - 1s - loss: 0.0292
Epoch 70/100
39/39 - 1s - loss: 0.0294
Epoch 71/100
39/39 - 1s - loss: 0.0315
Epoch 72/100
39/39 - 1s - loss: 0.0328
Epoch 73/100
39/39 - 1s - loss: 0.0335
Epoch 74/100
39/39 - 1s - loss: 0.0277
Epoch 75/100
```

```
39/39 - 1s - loss: 0.0334
Epoch 76/100
39/39 - 2s - loss: 0.0305
Epoch 77/100
39/39 - 1s - loss: 0.0312
Epoch 78/100
39/39 - 1s - loss: 0.0298
Epoch 79/100
39/39 - 1s - loss: 0.0295
Epoch 80/100
39/39 - 1s - loss: 0.0271
Epoch 81/100
39/39 - 1s - loss: 0.0329
Epoch 82/100
39/39 - 1s - loss: 0.0356
Epoch 83/100
39/39 - 1s - loss: 0.0332
Epoch 84/100
39/39 - 1s - loss: 0.0274
Epoch 85/100
39/39 - 1s - loss: 0.0319
Epoch 86/100
39/39 - 1s - loss: 0.0278
Epoch 87/100
39/39 - 1s - loss: 0.0307
Epoch 88/100
39/39 - 1s - loss: 0.0312
Epoch 89/100
39/39 - 1s - loss: 0.0286
Epoch 90/100
39/39 - 1s - loss: 0.0324
Epoch 91/100
39/39 - 1s - loss: 0.0295
Epoch 92/100
39/39 - 1s - loss: 0.0295
Epoch 93/100
39/39 - 1s - loss: 0.0303
Epoch 94/100
39/39 - 1s - loss: 0.0284
Epoch 95/100
39/39 - 1s - loss: 0.0280
Epoch 96/100
39/39 - 1s - loss: 0.0314
Epoch 97/100
39/39 - 1s - loss: 0.0294
Epoch 98/100
39/39 - 1s - loss: 0.0320
Epoch 99/100
```

```
39/39 - 1s - loss: 0.0334
Epoch 100/100
39/39 - 1s - loss: 0.0294
```

1.8.1 Plotting the training data model loss:

[536]:
```python
plt.plot(more_days.history['loss'],label='Training loss for more days')
plt.xlabel('Epochs')
plt.ylabel('Training loss')
plt.title('Network training Loss vs. Epochs( More days)')
plt.legend()
plt.show()
```