import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from sklearn.preprocessing import MinMaxScaler

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM

from tensorflow.keras.layers import Dense

from tensorflow.keras.layers import Dropout

In [ ]:

from google.colab import drive

drive.mount('/content/drive')

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force\_remount=True).

In [ ]:

from google.colab import files

uploaded = files.upload()

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

Saving Google\_Stock\_Price\_Test.csv to Google\_Stock\_Price\_Test.csv

Saving Google\_Stock\_Price\_Train.csv to Google\_Stock\_Price\_Train.csv

In [ ]:

dataset\_train = pd.read\_csv('Google\_Stock\_Price\_Train.csv')

dataset\_train.head()

--

*#keras only takes numpy array*

training\_set = dataset\_train.iloc[:, 1: 2].values

training\_set.shape

--

sc = MinMaxScaler(feature\_range = (0, 1))

*#fit: get min/max of train data*

training\_set\_scaled = sc.fit\_transform(training\_set)

In [ ]:

*#60 timesteps and 1 output*

X\_train = []

y\_train = []

for i in range(60, len(training\_set\_scaled)):

X\_train.append(training\_set\_scaled[i-60: i, 0])

y\_train.append(training\_set\_scaled[i, 0])

X\_train, y\_train = np.array(X\_train), np.array(y\_train)

In [ ]:

X\_train.shape

--

y\_train.shape

--

X\_train = np.reshape(X\_train, newshape = (X\_train.shape[0], X\_train.shape[1], 1))

In [ ]:

X\_train.shape

--

plt.figure(figsize=(18, 8))

plt.plot(dataset\_train['Open'])

plt.title("Google Stock Open Prices")

plt.xlabel("Time (oldest -> latest)")

plt.ylabel("Stock Open Price")

plt.show()

--

plt.figure(figsize=(18, 8))

plt.plot(dataset\_train['Low'])

plt.title("Google Stock Low Prices")

plt.xlabel("Time (oldest -> latest)")

plt.ylabel("Stock Lowest Price")

plt.show()

--

regressor = Sequential()

*#add 1st lstm layer*

regressor.add(LSTM(units = 50, return\_sequences = True, input\_shape = (X\_train.shape[1], 1)))

regressor.add(Dropout(rate = 0.2))

*##add 2nd lstm layer: 50 neurons*

regressor.add(LSTM(units = 50, return\_sequences = True))

regressor.add(Dropout(rate = 0.2))

*##add 3rd lstm layer*

regressor.add(LSTM(units = 50, return\_sequences = True))

regressor.add(Dropout(rate = 0.2))

*##add 4th lstm layer*

regressor.add(LSTM(units = 50, return\_sequences = False))

regressor.add(Dropout(rate = 0.2))

*##add output layer*

regressor.add(Dense(units = 1))

In [ ]:

regressor.compile(optimizer = 'adam', loss = 'mean\_squared\_error')

In [ ]:

regressor.fit(x = X\_train, y = y\_train, batch\_size = 32, epochs = 100)

--

dataset\_test = pd.read\_csv('Google\_Stock\_Price\_Test.csv')

dataset\_test.head()

--

*#keras only takes numpy array*

real\_stock\_price = dataset\_test.iloc[:, 1: 2].values

real\_stock\_price.shape

--

*#vertical concat use 0, horizontal uses 1*

dataset\_total = pd.concat((dataset\_train['Open'], dataset\_test['Open']),

axis = 0)

*#use .values to make numpy array*

inputs = dataset\_total[len(dataset\_total) - len(dataset\_test) - 60:].values

In [ ]:

*#reshape data to only have 1 col*

inputs = inputs.reshape(-1, 1)

*#scale input*

inputs = sc.transform(inputs)

len(inputs)

--

X\_test = []

for i in range(60, len(inputs)):

X\_test.append(inputs[i-60:i, 0])

X\_test = np.array(X\_test)

*#add dimension of indicator*

X\_test = np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], 1))

In [ ]:

X\_test.shape

--

predicted\_stock\_price = regressor.predict(X\_test)

--

*#inverse the scaled value*

predicted\_stock\_price = sc.inverse\_transform(predicted\_stock\_price)

In [ ]:

*#visualize the prediction and real price*

plt.plot(real\_stock\_price, color = 'red', label = 'Real price')

plt.plot(predicted\_stock\_price, color = 'blue', label = 'Predicted price')

plt.title('Google price prediction')

plt.xlabel('Time')

plt.ylabel('Price')

plt.legend()

plt.show()

--