**Module 11**

1.Write an LSTM program to predict next alphabet in the sequence “A B C D E F G H I J K L M N O P

Q R S T U V W X Y Z”

**Code:**

import numpy as np

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense

from tensorflow.keras.utils import to\_categorical

# Define the alphabet sequence

alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

# Create a mapping from characters to integers

char\_to\_int = {char: i for i, char in enumerate(alphabet)}

int\_to\_char = {i: char for i, char in enumerate(alphabet)}

# Prepare the dataset of input to output pairs encoded as integers

seq\_length = 1

dataX = []

dataY = []

for i in range(0, len(alphabet) - seq\_length):

seq\_in = alphabet[i:i + seq\_length]

seq\_out = alphabet[i + seq\_length]

dataX.append([char\_to\_int[char] for char in seq\_in])

dataY.append(char\_to\_int[seq\_out])

# Reshape X to be [samples, time steps, features]

X = np.reshape(dataX, (len(dataX), seq\_length, 1))

# Normalize the input

X = X / float(len(alphabet))

# One-hot encode the output variable

y = to\_categorical(dataY)

# Define the alphabet sequence

alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

# Create a mapping from characters to integers

char\_to\_int = {char: i for i, char in enumerate(alphabet)}

int\_to\_char = {i: char for i, char in enumerate(alphabet)}

# Prepare the dataset of input to output pairs encoded as integers

seq\_length = 1

dataX = []

dataY = []

for i in range(0, len(alphabet) - seq\_length):

seq\_in = alphabet[i:i + seq\_length]

seq\_out = alphabet[i + seq\_length]

dataX.append([char\_to\_int[char] for char in seq\_in])

dataY.append(char\_to\_int[seq\_out])

# Reshape X to be [samples, time steps, features]

X = np.reshape(dataX, (len(dataX), seq\_length, 1))

# Normalize the input

X = X / float(len(alphabet))

# One-hot encode the output variable

y = to\_categorical(dataY)

# Define a more complex LSTM model

model = Sequential()

model.add(LSTM(100, return\_sequences=True, input\_shape=(X.shape[1], X.shape[2])))

model.add(LSTM(50))

model.add(Dense(y.shape[1], activation='softmax'))

# Compile the model

model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

# Fit the model

model.fit(X, y, epochs=1000, batch\_size=10, verbose=2)

# Function to predict the next character

def predict\_next\_char(input\_char):

x = np.array([[char\_to\_int[input\_char]]]) / float(len(alphabet))

x = np.reshape(x, (1, seq\_length, 1))

prediction = model.predict(x, verbose=0)

index = np.argmax(prediction)

return int\_to\_char[index]

# Test the model

for char in alphabet:

next\_char = predict\_next\_char(char)

print(f"Input: {char}, Predicted Next Char: {next\_char}")

**Output:**

Epoch 1000/1000

3/3 - 0s - 18ms/step - accuracy: 1.0000 - loss: 0.3177

Input: A, Predicted Next Char: B

Input: B, Predicted Next Char: C

Input: C, Predicted Next Char: D

Input: D, Predicted Next Char: E

Input: E, Predicted Next Char: F

Input: F, Predicted Next Char: G

Input: G, Predicted Next Char: H

Input: H, Predicted Next Char: I

Input: I, Predicted Next Char: J

Input: J, Predicted Next Char: K

Input: K, Predicted Next Char: L

Input: L, Predicted Next Char: M

Input: M, Predicted Next Char: N

Input: N, Predicted Next Char: O

Input: O, Predicted Next Char: P

Input: P, Predicted Next Char: Q

Input: Q, Predicted Next Char: R

Input: R, Predicted Next Char: S

Input: S, Predicted Next Char: T

Input: T, Predicted Next Char: U

Input: U, Predicted Next Char: V

Input: V, Predicted Next Char: W

Input: W, Predicted Next Char: X

Input: X, Predicted Next Char: Y

Input: Y, Predicted Next Char: Z

Input: Z, Predicted Next Char: Z

2.Build a LSTM to predict the stock price for multivariate data(use multiple features).Use any online website or tiingo to get the dataset.

**Code:**

import numpy as np

import pandas as pd

from sklearn.preprocessing import MinMaxScaler

from sklearn.model\_selection import train\_test\_split

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dropout, Dense

from plotly.subplots import make\_subplots

import plotly.graph\_objects as go

import plotly.io as pi

import plotly.express as px

import datetime

from datetime import timedelta

stock\_data = pd.read\_csv('C:/Users/Lenovo/Downloads/Study material/AI/Recurrent Neural Network - LSTM and GRU/Assignment/Recurrent Neural network-LSTMs and GRUs/GOOG.csv')

fig = px.line(stock\_data, x='date', y='volume', title='Daily Trading Volume')

fig.update\_xaxes(title='Date')

fig.update\_yaxes(title='Volume')

fig.update\_layout(template='plotly\_dark')

fig.show()

fig = px.line(stock\_data, x='date', y='close', title='Closing Prices Over Time')

fig.update\_xaxes(title='Date')

fig.update\_yaxes(title='Closing Price')

fig.update\_layout(template='plotly\_dark')

fig.show()

fig.add\_trace(go.Scatter(x=stock\_data['date'], y=stock\_data['open'], mode='lines+markers', name='Open'))

fig.add\_trace(go.Scatter(x=stock\_data['date'], y=stock\_data['high'], mode='lines+markers', name='High'))

fig.add\_trace(go.Scatter(x=stock\_data['date'], y=stock\_data['low'], mode='lines+markers', name='Low'))

fig.add\_trace(go.Scatter(x=stock\_data['date'], y=stock\_data['close'], mode='lines+markers', name='Close'))

fig.update\_layout(title='Stock Price Analysis', xaxis\_title='Date', yaxis\_title='Price')

fig.show()

daily\_changes = stock\_data['close'].diff()

fig = px.histogram(daily\_changes, nbins=50, title='Histogram of Daily Price Changes')

fig.update\_xaxes(title='Daily Price Change')

fig.update\_yaxes(title='Frequency')

fig.update\_layout(template='plotly\_dark')

fig.show()

stock\_data['20-day MA'] = stock\_data['close'].rolling(window=20).mean()

fig = go.Figure(data=[go.Candlestick(

x=stock\_data['date'],

open=stock\_data['open'],

high=stock\_data['high'],

low=stock\_data['low'],

close=stock\_data['close'],

name="Candlesticks",

increasing\_line\_color='green',

decreasing\_line\_color='red',

line=dict(width=1),

showlegend=False

)])

fig.add\_trace(go.Scatter(x=stock\_data['date'], y=stock\_data['20-day MA'], mode='lines', name='20-day Moving Average', line=dict(color='rgba(255, 255, 0, 0.3)')))

fig.update\_layout(title="Google Stock Price Candlestick Chart with Moving Average", xaxis\_title="Date", yaxis\_title="Price", template="plotly\_dark")

fig.show()

stock\_data = stock\_data.drop('20-day MA', axis=1)

stock\_data['date'] = pd.to\_datetime(stock\_data['date'])

stock\_data = stock\_data.sort\_values('date')

stock = stock\_data[['date', 'close', 'high', 'low', 'open', 'volume']]

scaler = MinMaxScaler()

normalized\_data = stock[['open', 'high', 'low', 'volume', 'close']].copy()

normalized\_data = scaler.fit\_transform(normalized\_data)

train\_data, test\_data = train\_test\_split(normalized\_data, test\_size=0.2, shuffle=False)

train\_df = pd.DataFrame(train\_data, columns=['open', 'high', 'low', 'volume', 'close'])

test\_df = pd.DataFrame(test\_data, columns=['open', 'high', 'low', 'volume', 'close'])

def generate\_sequences(df, seq\_length=50):

X = df[['open', 'high', 'low', 'volume', 'close']].reset\_index(drop=True)

y = df[['open', 'high', 'low', 'volume', 'close']].reset\_index(drop=True)

sequences = []

labels = []

for index in range(len(X) - seq\_length + 1):

sequences.append(X.iloc[index : index + seq\_length].values)

labels.append(y.iloc[index + seq\_length - 1].values)

sequences = np.array(sequences)

labels = np.array(labels)

return sequences, labels

train\_sequences, train\_labels = generate\_sequences(train\_df)

test\_sequences, test\_labels = generate\_sequences(test\_df)

model = Sequential([

LSTM(units=50, return\_sequences=True, input\_shape=(50, 5)),

Dropout(0.2),

LSTM(units=50, return\_sequences=True),

Dropout(0.2),

LSTM(units=50),

Dropout(0.2),

Dense(units=5)

])

model.compile(loss='mean\_squared\_error', optimizer='adam', metrics=['mean\_absolute\_error'])

model.summary()

epochs = 200

batch\_size = 32

history = model.fit(train\_sequences, train\_labels, epochs=epochs, batch\_size=batch\_size, validation\_data=(test\_sequences, test\_labels), verbose=1)

train\_predictions = model.predict(train\_sequences)

test\_predictions = model.predict(test\_sequences)

fig = make\_subplots(rows=1, cols=1, subplot\_titles=('Close Predictions'))

train\_close\_pred = train\_predictions[:, 0]

train\_close\_actual = train\_labels[:, 0]

fig.add\_trace(go.Scatter(x=np.arange(len(train\_close\_actual)), y=train\_close\_actual, mode='lines', name='Actual', opacity=0.9))

fig.add\_trace(go.Scatter(x=np.arange(len(train\_close\_pred)), y=train\_close\_pred, mode='lines', name='Predicted', opacity=0.6))

fig.update\_layout(title='Close Predictions on Train Data', template='plotly\_dark')

fig.show()

latest\_prediction = []

last\_seq = test\_sequences[:-1]

for \_ in range(10):

prediction = model.predict(last\_seq)

latest\_prediction.append(prediction)

pi.templates.default = "plotly\_dark"

predicted\_data\_next = np.array(latest\_prediction).reshape(-1, 5)

last\_date = stock['date'].max()

next\_10\_days = [last\_date + timedelta(days=i) for i in range(1, 11)]

for i, feature\_name in enumerate(['open', 'high', 'low', 'volume', 'close']):

if feature\_name in ['volume', 'close']:

fig = go.Figure()

fig.add\_trace(go.Scatter(x=next\_10\_days, y=predicted\_data\_next[:, i], mode='lines+markers', name=f'Predicted {feature\_name.capitalize()} Prices'))

fig.update\_layout(title=f'Predicted {feature\_name.capitalize()} Prices for the Next 10 Days', xaxis\_title='Date', yaxis\_title=f'{feature\_name.capitalize()} Price')

fig.show()

**Output:**



