**#RNN**

import tensorflow as tf

from tensorflow.keras.datasets import imdb

from tensorflow.keras.preprocessing.sequence import pad\_sequences

import matplotlib.pyplot as plt

from sklearn.metrics import confusion\_matrix, accuracy\_score

# Load and preprocess dataset

(x\_train, y\_train), (x\_test, y\_test) = imdb.load\_data(num\_words=20000)

x\_train, x\_test = pad\_sequences(x\_train, maxlen=100), pad\_sequences(x\_test, maxlen=100)

# Build and compile model

model = tf.keras.Sequential([

tf.keras.layers.Embedding(input\_dim=20000, output\_dim=128, input\_length=100),

tf.keras.layers.LSTM(units=128, activation='tanh'),

tf.keras.layers.Dense(units=1, activation='sigmoid')

])

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

# Train model

history = model.fit(x\_train, y\_train, batch\_size=128, epochs=5, validation\_data=(x\_test, y\_test))

# Predictions and evaluation

y\_pred = (model.predict(x\_test) > 0.5).astype("int32")

print(confusion\_matrix(y\_test, y\_pred))

print(accuracy\_score(y\_test, y\_pred))

# Plot learning curve

epochs = range(1, 6)

plt.plot(epochs, history.history['accuracy'], label='Train')

plt.plot(epochs, history.history['val\_accuracy'], label='Validation')

plt.title('Model Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

plt.plot(epochs, history.history['loss'], label='Train')

plt.plot(epochs, history.history['val\_loss'], label='Validation')

plt.title('Model Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend()

plt.show()

#CNN

import tensorflow as tf

import numpy as np

import matplotlib.pyplot as plt

from sklearn.metrics import confusion\_matrix, accuracy\_score

from tensorflow.keras.datasets import mnist

# Load and preprocess dataset

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

x\_train, x\_test = x\_train / 255.0, x\_test / 255.0

x\_train = x\_train.reshape(-1, 28, 28, 1)

x\_test = x\_test.reshape(-1, 28, 28, 1)

# Build and compile model

model = tf.keras.Sequential([

tf.keras.Input(shape=(28, 28, 1)), # Explicit Input layer

tf.keras.layers.Conv2D(32, (3,3), activation='relu'),

tf.keras.layers.Conv2D(64, (3,3), activation='relu'),

tf.keras.layers.MaxPool2D((2,2)),

tf.keras.layers.Dropout(0.4),

tf.keras.layers.Flatten(),

tf.keras.layers.Dense(128, activation='relu'),

tf.keras.layers.Dense(10, activation='softmax')

])

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['sparse\_categorical\_accuracy'])

# Train model

history = model.fit(x\_train, y\_train, batch\_size=128, epochs=10, validation\_data=(x\_test, y\_test))

# Model predictions and evaluation

y\_pred = model.predict(x\_test).argmax(axis=1)

print(confusion\_matrix(y\_test, y\_pred))

print(accuracy\_score(y\_test, y\_pred))

# Plot learning curves

epochs = range(1, 11)

plt.plot(epochs, history.history['sparse\_categorical\_accuracy'], label='Train')

plt.plot(epochs, history.history['val\_sparse\_categorical\_accuracy'], label='Validation')

plt.title('Model Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

plt.plot(epochs, history.history['loss'], label='Train')

plt.plot(epochs, history.history['val\_loss'], label='Validation')

plt.title('Model Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend()

plt.show()

**GOOGLE PRICE PREDICTOR**

import tensorflow as tf

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

# Read datasets from local directory

train\_data = pd.read\_csv(r"C:\Users\Admin\Desktop\Google\_Stock\_Price\_Train.csv")

test\_data = pd.read\_csv(r"C:\Users\Admin\Desktop\Google\_Stock\_Price\_Test.csv")

# Preprocess training data

train\_set = train\_data.iloc[:, 1:2].values

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

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

# Create training sequences

x\_train, y\_train = [], []

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

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

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

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

x\_train = x\_train.reshape(x\_train.shape[0], x\_train.shape[1], 1)

# Build LSTM model

model = tf.keras.models.Sequential([

tf.keras.layers.LSTM(60, activation='relu', return\_sequences=True,),

tf.keras.layers.Dropout(0.2),

tf.keras.layers.LSTM(60, activation='relu', return\_sequences=True),

tf.keras.layers.Dropout(0.2),

tf.keras.layers.LSTM(80, activation='relu', return\_sequences=True),

tf.keras.layers.Dropout(0.2),

tf.keras.layers.LSTM(120, activation='relu'),

tf.keras.layers.Dropout(0.2),

tf.keras.layers.Dense(1)

])

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

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

# Preprocess test data

real\_stock\_price = test\_data.iloc[:, 1:2].values

total\_data = pd.concat((train\_data['Open'], test\_data['Open']), axis=0)

inputs = total\_data[len(total\_data) - len(test\_data) - 60:].values.reshape(-1, 1)

inputs = sc.transform(inputs)

# Create test sequences

x\_test = np.array([inputs[i-60:i, 0] for i in range(60, 80)])

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

# Predict stock prices

predicted\_stock\_price = model.predict(x\_test)

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

# Visualization

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

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

plt.title('Google Stock Price Prediction')

plt.xlabel('Time')

plt.ylabel('Google Stock Price')

plt.legend()

plt.show()

KMEANS

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

data = list(zip([4, 5, 10, 4, 3, 11, 14, 6, 10, 12], [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]))

inertias = [KMeans(n\_clusters=i).fit(data).inertia\_ for i in range(1, 11)]

plt.plot(range(1, 11), inertias)

plt.title('Elbow method')

plt.xlabel('Number of clusters')

plt.ylabel('Inertia')

plt.show()

kmeans = KMeans(n\_clusters=2).fit(data)

plt.scatter(\*zip(\*data), c=kmeans.labels\_)

plt.show()

Neural Network

import numpy as np

'

'

# Define the input data and target output

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)

y = np.array(([92], [86], [89]), dtype=float)

X = X / np.amax(X, axis=0)

y = y / 100

# Activation function (sigmoid) and its derivative

def sigmoid(x):

return 1 / (1 + np.exp(-x))

def derivatives\_sigmoid(x):

return x \* (1 - x)

# Neural network parameters

epoch = 5

lr = 0.1

inputlayer\_neurons = 2

hiddenlayer\_neurons = 3

output\_neurons = 1

# Weight and bias initialization

wh = np.random.uniform(size=(inputlayer\_neurons, hiddenlayer\_neurons))

bh = np.random.uniform(size=(1, hiddenlayer\_neurons))

wout = np.random.uniform(size=(hiddenlayer\_neurons, output\_neurons))

bout = np.random.uniform(size=(1, output\_neurons))

# Training loop

for i in range(epoch):

# Forward Propagation

hinp = np.dot(X, wh) + bh

hlayer\_act = sigmoid(hinp)

outinp = np.dot(hlayer\_act, wout) + bout

output = sigmoid(outinp)

# Backpropagation

EO = y - output

outgrad = derivatives\_sigmoid(output)

d\_output = EO \* outgrad

EH = d\_output.dot(wout.T)

hiddengrad = derivatives\_sigmoid(hlayer\_act)

d\_hiddenlayer = EH \* hiddengrad

# Update weights and biases

wout += hlayer\_act.T.dot(d\_output) \* lr

wh += X.T.dot(d\_hiddenlayer) \* lr

# Print epoch results

print(f"-----------Epoch-{i+1} Starts----------")

print("Input:\n", X)

print("Actual Output:\n", y)

print("Predicted Output:\n", output)

print(f"-----------Epoch-{i+1} Ends----------\n")

# Final prediction after training

print("Input:\n", X)

print("Actual Output:\n", y)

print("Predicted Output:\n", output)