LAB PROGRAM 1

import tensorflow as tf

scalar=tf.constant(6)

print("scalar: \n",scalar)

print("\n")

#1D tensor

vector=tf.constant([1,2,3])

print("vector: \n",vector)

print("\n")

#2D tensor

matrix=tf.constant([[1,2],[3,4]])

print("matrix: \n",matrix)

print("\n")

#3D tensor

tensor=tf.constant([[[1,2],[3,4]],[[5,6],[7,8]]])

print("tensor:\n",tensor)

print("\n")

#addition of tensor

tensor\_a=tf.constant([[1,2],[3,4]])

tensor\_b=tf.constant([[1,2],[3,4]])

addition=tf.add(tensor\_a,tensor\_b)

print("addition: \n",addition)

print("\n")

#matrix-multiplication

tensor\_a=tf.constant([[1,2],[3,4]])

tensor\_b=tf.constant([[1,2],[3,4]])

matrixmultiplication=tf.matmul(tensor\_a,tensor\_b)

print("matrix multiplication: \n",matrixmultiplication)

print("\n")

#reshape

tensor\_a=tf.constant([[1,2],[3,4],[5,6]])

reshape=tf.reshape(tensor\_a,[2,3])

print("reshape:\n",reshape)

print("\n")

#element-wise multiplication

tensor\_a=tf.constant([[1,2],[3,4]])

tensor\_b=tf.constant([[1,2],[3,4]])

elementwisemultiplication=tf.multiply(tensor\_a,tensor\_b)

print("elementwise multiplication: \n",elementwisemultiplication)

print("\n")

#sequential model

model=tf.keras.Sequential([

    tf.keras.layers.Dense(64,activation="relu",input\_shape=(10,1)),

    tf.keras.layers.Dense(64,activation="relu"),

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

])

model.summary()

LAB PROGRAM 2

import tensorflow as tf

from tensorflow import keras

from keras import layers

import numpy as np

import matplotlib.pyplot as plt

#load the data

cifar100=tf.keras.datasets.cifar100

#distribute th data into train and validation

(x\_train,y\_train),(x\_val,y\_val)=cifar100.load\_data()

print(x\_train.shape,y\_train.shape)

print(x\_val.shape,y\_val.shape)

import warnings

warnings.filterwarnings('ignore')

y\_train=tf.one\_hot(y\_train,depth=100,dtype=tf.float64)

y\_val=tf.one\_hot(y\_val,depth=100,dtype=tf.float64)

y\_train=tf.squeeze(y\_train)

y\_val=tf.squeeze(y\_val)

model=tf.keras.models.Sequential([

   layers.Conv2D(16,(3,3),activation="relu",padding="same",input\_shape=(32,32,3)),

   layers.Conv2D(32,(3,3),activation="relu",padding="same"),

   layers.Conv2D(64,(3,3),activation="relu",padding="same"),

   layers.MaxPooling2D(2,2),

   layers.Conv2D(128,(3,3),activation="relu",padding="same"),

   layers.Flatten(),

   layers.Dense(256,activation="relu"),

   layers.BatchNormalization(),

   layers.Dense(256,activation="relu"),

   layers.Dropout(0.3),

   layers.BatchNormalization(),

   layers.Dense(100,activation="softmax")

])

model.summary()

model.compile(

    loss=tf.keras.losses.CategoricalCrossentropy(from\_logits=False),

    optimizer="Adam",

    metrics=["accuracy"]

)

history=model.fit(

    x\_train,y\_train,

    epochs=5,

    validation\_data=(x\_val,y\_val),

    batch\_size=100,

    verbose=1

)

loss,accuracy=model.evaluate(x\_val,y\_val)

print("loss: ",loss)

print("accuarcy: ",accuracy)

LAB PROGRAM 6

import tensorflow as tf

import numpy as np

import matplotlib.pyplot as plt

from tensorflow.keras import layers

from tensorflow.keras.layers import Input,Conv2D,UpSampling2D,MaxPooling2D

from tensorflow.keras.datasets import mnist

from tensorflow.keras.models import Model

#load the data

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

#normalize the data

x\_train=x\_train.astype('float32')/255.

x\_test=x\_test.astype('float32')/255.

#reshape to 28,28

x\_train=np.reshape(x\_train,(len(x\_train),28,28,1))

x\_test=np.reshape(x\_test,(len(x\_test),28,28,1))

#add the noise to data

noise\_factor=0.5

x\_train\_noisy=x\_train+noise\_factor\*np.random.normal(loc=0.0,scale=1.0,size=x\_train.shape)

x\_test\_noisy=x\_test+noise\_factor\*np.random.normal(loc=0.0,scale=1.0,size=x\_test.shape)

#build the model

input\_img=Input(shape=(28,28,1))

x=Conv2D(32,(3,3),activation="relu",padding="same")(input\_img)

x=MaxPooling2D((2,2),padding="same")(x)

x=Conv2D(32,(3,3),activation="relu",padding="same")(x)

encoder=MaxPooling2D((2,2),padding="same")(x)

x=Conv2D(32,(3,3),activation="relu",padding="same")(encoder)

x=UpSampling2D((2,2))(x)

x=Conv2D(32,(3,3),activation="relu",padding="same")(x)

x=UpSampling2D((2,2))(x)

decoder=Conv2D(1,(3,3),activation="sigmoid",padding="same")(x)

#compile

autoencoder=Model(input\_img,decoder)

autoencoder.compile(optimizer="adam",loss="binary\_crossentropy")

#train

autoencoder.fit(x\_train\_noisy,x\_train,epochs=5,shuffle=True,validation\_data=(x\_test\_noisy,x\_test),batch\_size=128)

#predict

decoded\_img=autoencoder.predict(x\_test\_noisy)

#plot the graph

n=10

plt.figure(figsize=(20,4))

for i in range(n):

  ax=plt.subplot(3,n,i+1)

  plt.imshow(x\_test\_noisy[i].reshape(28,28),cmap="gray")

  ax.axis("off")

  ax=plt.subplot(3,n,i+n+1)

  plt.imshow(decoded\_img[i].reshape(28,28),cmap="gray")

  ax.axis("off")

  ax=plt.subplot(3,n,i+2\*n+1)

  plt.imshow(x\_test[i].reshape(28,28),cmap="gray")

  ax.axis("off")

plt.tight\_layout()

plt.show()

import numpy as np

import pandas as pd

import tensorflow as tf

LAB PROGRAM 7

import matplotlib.pyplot as plt

url="https://raw.githubusercontent.com/jbrownlee/Datasets/master/daily-min-temperatures.csv"

df=pd.read\_csv(url,parse\_dates=["Date"])

print(df)

plt.figure(figsize=(10,4))

plt.plot(df["Temp"])

plt.title("Daily minimum temperature")

plt.xlabel("Dates")

plt.ylabel("temperature")

plt.show()

temps=df["Temp"].values.astype(np.float32)

mean1=temps.mean()

std1=temps.std()

temps=(temps-mean1)/std1

def create\_squence(data,window\_size):

  X,y=[],[]

  for i in range(len(data)-window\_size):

    X.append(data[i:i+window\_size])

    y.append(data[i+window\_size])

  return np.array(X),np.array(y)

window\_size=7

X,y=create\_squence(temps,window\_size)

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

split=int(0.8\*len(X))

x\_train,x\_test=X[:split],X[split:]

y\_train,y\_test=y[:split],y[split:]

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense,SimpleRNN

model=Sequential([

    SimpleRNN(64,return\_sequences=True,activation="tanh",input\_shape=(window\_size,1)),

    SimpleRNN(32,activation="tanh"),

    Dense(1)

])

model.compile(optimizer="adam",loss="mse")

model.summary()

from tensorflow.keras.callbacks import EarlyStopping

early\_stop=EarlyStopping(monitor="val\_loss",patience=5,restore\_best\_weights=True)

history=model.fit(

    x\_train,y\_train,

    validation\_data=(x\_test,y\_test),

    callbacks=[early\_stop],

    batch\_size=32,

    epochs=100

)

predictions=model.predict(x\_test)

predicted\_values=predictions\*std1+mean1

actual\_values=y\_test\*std1+mean1

plt.figure(figsize=(20,4))

plt.plot(predicted\_values,label="predicted")

plt.plot(actual\_values,label="actual")

plt.legend()

plt.title("---")

plt.show()

LAB PROGRAM 8

import numpy as np

import pandas as pd

import tensorflow as tf

import matplotlib.pyplot as plt

url="https://raw.githubusercontent.com/jbrownlee/Datasets/master/daily-min-temperatures.csv"

df=pd.read\_csv(url,parse\_dates=["Date"])

print(df)

plt.figure(figsize=(10,4))

plt.plot(df["Temp"])

plt.title("Daily minimum temperature")

plt.xlabel("Dates")

plt.ylabel("temperature")

plt.show()

temps=df["Temp"].values.astype(np.float32)

mean1=temps.mean()

std1=temps.std()

temps=(temps-mean1)/std1

def create\_squence(data,window\_size):

  X,y=[],[]

  for i in range(len(data)-window\_size):

    X.append(data[i:i+window\_size])

    y.append(data[i+window\_size])

  return np.array(X),np.array(y)

window\_size=7

X,y=create\_squence(temps,window\_size)

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

split=int(0.8\*len(X))

x\_train,x\_test=X[:split],X[split:]

y\_train,y\_test=y[:split],y[split:]

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense,LSTM

model=Sequential([

    LSTM(64,return\_sequences=True,activation="tanh",input\_shape=(window\_size,1)),

    LSTM(32,activation="tanh"),

    Dense(1)

])

model.compile(optimizer="adam",loss="mse")

model.summary()

from tensorflow.keras.callbacks import EarlyStopping

early\_stop=EarlyStopping(monitor="val\_loss",patience=5,restore\_best\_weights=True)

history=model.fit(

    x\_train,y\_train,

    validation\_data=(x\_test,y\_test),

    callbacks=[early\_stop],

    batch\_size=32,

    epochs=100

)

predictions=model.predict(x\_test)

predicted\_values=predictions\*std1+mean1

actual\_values=y\_test\*std1+mean1

plt.figure(figsize=(10,4))

plt.plot(predicted\_values,label="predicted")

plt.plot(actual\_values,label="actual")

plt.legend()

plt.title("---")

plt.show()