EXP 1 - Implement logic gates using single layer perceptron

import numpy as np

def unitStep(v):

if v>=0:

return 1

else:

return 0

def perceptronModel(x, w, b):

v = np.dot(w, x)+ b

y = unitStep(v)

return y

def OR\_logicFunction(x):

w = np.array([1,1])

b =-0.5

return perceptronModel(x, w,b)

test1= np.array([0,1])

test2= np.array([1,1])

test3= np.array([0,0])

test4= np.array([1,0])

print("OR({},{}) = {}".format(0,1, OR\_logicFunction(test1)))

print("OR({},{}) = {}".format(1,1, OR\_logicFunction(test2)))

print("OR({},{}) = {}".format(0,0, OR\_logicFunction(test3)))

print("OR({},{}) = {}".format(1,0, OR\_logicFunction(test4)))

EXP 2 - Implement a two class neural network with a hidden layer on any standard dataset.

# importing modules

import tensorflow as tf

import numpy as np

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Flatten

from tensorflow.keras.layers import Dense

from tensorflow.keras.layers import Activation

import matplotlib.pyplot as plt

(x\_train, y\_train), (x\_test, y\_test) = tf.keras.datasets.mnist.load\_data(path = '/Users//Downloads/mnist.npz')

# Cast the records into float values

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

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

# normalize image pixel values by dividing

# by 255 gray\_scale

gray\_scale= 255

x\_train /= gray\_scale

x\_test /= gray\_scale

print("Feature matrix:", x\_train.shape)

print("Target matrix:", x\_test.shape)

print("Feature matrix:", y\_train.shape)

print("Target matrix:", y\_test.shape)

fig, ax = plt.subplots(10, 10)

k = 0

for i in range(10):

for j in range(10):

ax[i][j].imshow(x\_train[k].reshape(28, 28),

aspect='auto')

k += 1

plt.show()

model = Sequential([

# reshape 28 row \* 28 column data to 28\*28 rows

Flatten(input\_shape=(28, 28)),

# dense layer 1

Dense(256, activation='sigmoid'),

# dense layer 2

Dense(128, activation='sigmoid'),

# output layer

Dense(10, activation='sigmoid'),

])

model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

model.fit(x\_train, y\_train, epochs=10,

batch\_size=2000,

validation\_split=0.2)

results = model.evaluate(x\_test, y\_test, verbose = 0)

print('test loss, test acc:', results)

EXP 3 - Using CNN and MNIST dataset, perform digit classification

import keras

from keras.datasets import mnist

from keras.models import Sequential *# Import Sequential*

from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense *# Import necessary layers*

import matplotlib.pyplot as plt

import numpy as np

from tensorflow.keras.datasets import mnist

# Load dataset from local file

(X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data(path='/Users/ /Downloads/mnist.npz')

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

fig = plt.figure()

for i in range(9):

plt.subplot(3, 3, i+1)

plt.tight\_layout()

plt.imshow(X\_train[i], cmap='gray', interpolation='none')

plt.title("Digit: {}".format(y\_train[i]))

plt.xticks([])

plt.yticks([])

plt.show()

#reshaping

#this assumes our data format

#For 3D data,"channels\_last" assumes (conv\_dim1, conv\_dim2, conv\_dim3, channels) while

#"channels\_first" assumes (channels, conv\_dim1, conv\_dim2, conv\_dim3).

img\_rows, img\_cols=28, 28

if keras.backend.image\_data\_format() == 'channels\_first':

X\_train = X\_train.reshape(X\_train.shape[0], 1, img\_rows, img\_cols)

X\_test = X\_test.reshape(X\_test.shape[0], 1, img\_rows, img\_cols)

input\_shape = (1, img\_rows, img\_cols)

else:

X\_train = X\_train.reshape(X\_train.shape[0], img\_rows, img\_cols, 1)

X\_test = X\_test.reshape(X\_test.shape[0], img\_rows, img\_cols, 1)

input\_shape = (img\_rows, img\_cols, 1)

#more reshaping

X\_train = X\_train.astype('float32')

X\_test = X\_test.astype('float32')

X\_train /= 255

X\_test /= 255

print('X\_train shape:', X\_train.shape) *#X\_train shape: (60000, 28, 28, 1)*

import keras

#set number of categories

num\_category = 10

# convert class vectors to binary class matrices

y\_train = keras.utils.to\_categorical(y\_train, num\_category)

y\_test = keras.utils.to\_categorical(y\_test, num\_category)

##model building

model = Sequential()

#convolutional layer with rectified linear unit activation

model.add(Conv2D(32, kernel\_size=(3, 3),

activation='relu',input\_shape=input\_shape))

#32 convolution filters used each of size 3x3

#again

model.add(Conv2D(64, (3, 3), activation='relu'))

#64 convolution filters used each of size 3x3

#choose the best features via pooling

model.add(MaxPooling2D(pool\_size=(2, 2)))

#randomly turn neurons on and off to improve convergence

model.add(Dropout(0.25))

#flatten since too many dimensions, we only want a classification output

model.add(Flatten())

#fully connected to get all relevant data

model.add(Dense(128, activation='relu'))

#one more dropout for convergence' sake :)

model.add(Dropout(0.5))

#output a softmax to squash the matrix into output probabilities

model.add(Dense(num\_category, activation='softmax'))

#Adaptive learning rate (adaDelta) is a popular form of gradient descent rivaled only by adam and adagrad

#categorical ce since we have multiple classes (10)

model.compile(loss=keras.losses.categorical\_crossentropy,optimizer=keras.optimizers.Adadelta(),metrics=['accuracy'])

batch\_size = 128

num\_epoch = 10

#model training

model\_log = model.fit(X\_train, y\_train,

batch\_size=batch\_size,

epochs=num\_epoch,

verbose=1,

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

score = model.evaluate(X\_test, y\_test, verbose=0)

print('Test loss:', score[0]) *#Test loss: 0.0296396646054*

print('Test accuracy:', score[1]) *#Test accuracy: 0.9904*

fig=plt.figure()

plt.subplot(2, 1, 1)

plt.plot(model\_log.history['accuracy']) *# Updated to 'accuracy'*

plt.plot(model\_log.history['val\_accuracy']) *# Updated to 'val\_accuracy'*

plt.title('Model Accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Test'], loc='lower right')

plt.subplot(2, 1, 2)

plt.plot(model\_log.history['loss'])

plt.plot(model\_log.history['val\_loss'])

plt.title('Model Loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Train', 'Test'], loc='upper right')

plt.tight\_layout()

plt.show()

model\_digit\_json=model.to\_json()

with open("model\_digit.json", "w") as json\_file:

json\_file.write(model\_digit\_json)

# Serialize weights to HDF5

model.save\_weights("model\_digit.weights.h5") *# Corrected filename*

print("Saved model to disk")

EXP 4 - Develop an Autoencoder forhandwritten digits dataset MNIST.

from tensorflow.keras.datasets import mnist

from tensorflow.keras.layers import Dense, Input, Flatten, Reshape,LeakyReLU as LR, Activation, Dropout

from tensorflow.keras.models import Model, Sequential

from matplotlib import pyplot as plt

from IPython import display # If using IPython, Colab, or Jupyter

import numpy as np

# Load MNIST data

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data(path='/Users/ /Downloads/mnist.npz')

x\_train = x\_train / 255.0

x\_test = x\_test / 255.0

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

# x\_train = x\_train / 255.0

# x\_test = x\_test / 255.0

# Plot image data from x\_train

plt.imshow(x\_train[0], cmap = "gray")

plt.show()

LATENT\_SIZE = 32

#Creating the Encoder

encoder = Sequential([

Flatten(input\_shape=(28, 28)),

Dense(512),

LR(),

Dropout(0.5),

Dense(256),

LR(),

Dropout(0.5),

Dense(128),

LR(),

Dropout(0.5),

Dense(64),

LR(),

Dropout(0.5),

Dense(LATENT\_SIZE),

LR()

])

# Define decoder model

decoder = Sequential([

Dense(64, input\_shape=(LATENT\_SIZE,)),

LR(),

Dropout(0.5),

Dense(128),

LR(),

Dropout(0.5),

Dense(256),

LR(),

Dropout(0.5),

Dense(512),

LR(),

Dropout(0.5),

Dense(784),

Activation("sigmoid"),

Reshape((28, 28))

])

# Build autoencoder model

img = Input(shape=(28, 28))

latent\_vector = encoder(img)

output = decoder(latent\_vector)

model = Model(inputs=img, outputs=output)

# Compile the model

model.compile(optimizer="nadam", loss="binary\_crossentropy")

EPOCHS = 60

# Training and visualization loop

for epoch in range(EPOCHS):

fig, axs = plt.subplots(4, 4)

rand = x\_test[np.random.randint(0, 10000, 16)].reshape((16, 28,28))

display.clear\_output(wait=True) # If you imported display fromIPython

for i in range(4):

for j in range(4):

axs[i, j].imshow(model.predict(rand[i \* 4 + j].reshape(1,28, 28))[0], cmap="gray")

axs[i, j].axis("off")

plt.subplots\_adjust(wspace=0, hspace=0)

plt.show()

print("-----------", "EPOCH", epoch + 1,"-----------")

# Train the model

model.fit(x\_train, x\_train, epochs=1, verbose=1)

EXP 5 - Using IMDB review dataset, perform sentiment classification using LSTM and BiLSTM and compare result

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from tensorflow.keras.datasets import imdb

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

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, LSTM, Dense, Bidirectional

from tensorflow.keras.utils import to\_categorical

(train\_data, train\_labels), (test\_data, test\_labels) = imdb.load\_data(path='/Users/ /Downloads/imdb.npz', num\_words=10000)

print(f'Training data shape: {train\_data.shape}')

print(f'Training labels shape: {train\_labels.shape}')

print(f'Testing data shape: {test\_data.shape}')

print(f'Testing labels shape: {test\_labels.shape}')

max\_length = 500

train\_data = pad\_sequences(train\_data, maxlen=max\_length)

test\_data = pad\_sequences(test\_data, maxlen=max\_length)

# Convert labels to categorical (if necessary)

train\_labels = np.array(train\_labels)

test\_labels = np.array(test\_labels)

# Check padded data shape

print(f'Padded training data shape: {train\_data.shape}')

print(f'Padded testing data shape: {test\_data.shape}')

def create\_model(is\_bidirectional=False):

model = Sequential()

model.add(Embedding(input\_dim=10000, output\_dim=128, input\_length=max\_length))

if is\_bidirectional:

model.add(Bidirectional(LSTM(64)))

else:

model.add(LSTM(64))

model.add(Dense(1, activation='sigmoid')) *# Binary classification*

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

return model

# Create LSTM model

lstm\_model = create\_model()

lstm\_model.summary()

lstm\_history = lstm\_model.fit(

train\_data,

train\_labels,

epochs=5,

batch\_size=64,

validation\_split=0.2,

verbose=1 # Set verbosity level (0 for silent, 1 for progress bar, 2 for one line per epoch)

)

lstm\_loss, lstm\_accuracy = lstm\_model.evaluate(test\_data, test\_labels)

print(f'LSTM Test Accuracy: {lstm\_accuracy:.2f}')

# Create BiLSTM model

bilstm\_model = create\_model(is\_bidirectional=True)

bilstm\_model.summary()

# Train BiLSTM model

bilstm\_history = bilstm\_model.fit(

train\_data,

train\_labels,

epochs=5,

batch\_size=64,

validation\_split=0.2, *# Added validation\_split*

verbose=1 # Optional, to show training progress

)

# Evaluate BiLSTM model

bilstm\_loss, bilstm\_accuracy = bilstm\_model.evaluate(test\_data, test\_labels)

print(f'BiLSTM Test Accuracy: {bilstm\_accuracy:.2f}')

# Plot training & validation accuracy

plt.plot(lstm\_history.history['accuracy'], label='LSTM Training Accuracy')

plt.plot(lstm\_history.history['val\_accuracy'], label='LSTM Validation Accuracy')

plt.plot(bilstm\_history.history['accuracy'], label='BiLSTM Training Accuracy')

plt.plot(bilstm\_history.history['val\_accuracy'], label='BiLSTM Validation Accuracy')

plt.title('Model Accuracy Comparison')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

EXP 6 - Using IMDB review dataset,perform sentiment classification using RNN.

from keras.datasets import imdb

from keras.preprocessing import sequence

from keras.models import Sequential

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

vocabulary\_size = 5000

(X\_train, y\_train), (X\_test, y\_test) =imdb.load\_data(path = '/Users/ /Downloads/imdb.npz',num\_words=vocabulary\_size)

print('Loaded dataset with {} training samples, {} test samples'.format(len(X\_train), len(X\_test)))

# Map review back to original words

word2id = imdb.get\_word\_index(path ='/Users/ /Downloads/imdb\_word\_index.json')

id2word = {i: word for word, i in word2id.items()}

print('---review with words---')

print([id2word.get(i, ' ') for i in X\_train[6]])

print('---label---')

print(y\_train[6])

# Maximum and minimum review lengths

print('Maximum review length: {}'

.format(len(max((X\_train + X\_test),

key=len))))

print('Minimum review length: {}'

.format(len(min((X\_train + X\_test),

key=len))))

# Pad sequences

max\_words = 500

X\_train = sequence.pad\_sequences(X\_train, maxlen=max\_words)

X\_test = sequence.pad\_sequences(X\_test, maxlen=max\_words)

# Design RNN model

model = Sequential()

embedding\_size = 32

model.add(Embedding(vocabulary\_size, embedding\_size,

input\_length=max\_words))

model.add(LSTM(100))

model.add(Dense(1, activation='sigmoid'))

# Model summary

print(model.summary())

# Compile model

model.compile(loss='binary\_crossentropy', optimizer='adam',

metrics=['accuracy'])

# Train model

batch\_size = 64

num\_epochs = 3

X\_valid, y\_valid = X\_train[:batch\_size], y\_train[:batch\_size]

X\_train2, y\_train2 = X\_train[batch\_size:], y\_train[batch\_size:]

model.fit(X\_train2, y\_train2, validation\_data=(X\_valid, y\_valid),

batch\_size=batch\_size, epochs=num\_epochs)

# Evaluate model

scores = model.evaluate(X\_test, y\_test, verbose=0)

print('Test accuracy:', scores[1])