from google.colab import drive

drive.mount('/content/gdrive')

path\_to\_csv = '/content/gdrive/MyDrive/Colab Notebooks/diabetes.csv'

pip install tensorflow

import keras

import pandas

import tensorflow as tf

from tensorflow.keras.layers import Dense, Activation  # You can import individual layers like Dense from keras.layers

from keras.models import Sequential

# load dataset

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

import pandas as pd

import numpy as np

dataset = pd.read\_csv(path\_to\_csv, header=None).values

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(dataset[:,0:8], dataset[:,8],

                                                    test\_size=0.25, random\_state=87)

np.random.seed(155)

my\_first\_nn = Sequential() # create model

my\_first\_nn.add(Dense(20, input\_dim=8, activation='relu')) # hidden layer

my\_first\_nn.add(Dense(1, activation='sigmoid')) # output layer

my\_first\_nn.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['acc'])

my\_first\_nn\_fitted = my\_first\_nn.fit(X\_train, Y\_train, epochs=100,

                                     initial\_epoch=0)

print(my\_first\_nn.summary())

print(my\_first\_nn.evaluate(X\_test, Y\_test))

A screenshot of a computer

Description automatically generated

import keras

import pandas as pd

import numpy as np

from keras.models import Sequential

import tensorflow as tf

from tensorflow.keras.layers import Dense, Activation  # You can import individual layers like Dense from keras.layers

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

# load dataset

path\_to\_csv = '/content/gdrive/MyDrive/Colab Notebooks/diabetes.csv'

dataset = pd.read\_csv(path\_to\_csv, header=None).values

# split dataset into training and test sets

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(dataset[:,0:8], dataset[:,8],

                                                    test\_size=0.25, random\_state=87)

# define the model

np.random.seed(155)

my\_second\_nn = Sequential()

my\_second\_nn.add(Dense(20, input\_dim=8, activation='relu'))

my\_second\_nn.add(Dense(20, input\_dim=8,activation='relu'))

my\_second\_nn.add(Dense(20, input\_dim=8,activation='relu'))

my\_second\_nn.add(Dense(1, activation='sigmoid'))

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

# train the model

my\_second\_nn\_fitted= my\_second\_nn.fit(X\_train, Y\_train, epochs=100,

                                     initial\_epoch=0)

# evaluate the model on the test set

score = my\_second\_nn.evaluate(X\_test, Y\_test, batch\_size=64)

print(my\_second\_nn.summary())

print("Test accuracy:", score[1])

A screenshot of a computer

Description automatically generated

path\_to\_csv = '/content/gdrive/MyDrive/Colab Notebooks/breastcancer.csv'

import pandas as pd

import numpy as np

from sklearn.datasets import load\_breast\_cancer

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

from sklearn.preprocessing import StandardScaler

from keras.models import Sequential

from keras.layers import Dense

# Load dataset

data = load\_breast\_cancer()

# Split dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data.data, data.target,

                                                    test\_size=0.25, random\_state=87)

# Normalize data

sc = StandardScaler()

X\_train\_norm = sc.fit\_transform(X\_train)

X\_test\_norm = sc.transform(X\_test)

# Create model

np.random.seed(155)

model = Sequential()

model.add(Dense(20, input\_dim=30, activation='relu'))

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

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

# Train model

model.fit(X\_train\_norm, y\_train, epochs=100, initial\_epoch=0)

# Evaluate model on testing set

loss, accuracy = model.evaluate(X\_test\_norm, y\_test)

print(model.summary())

print("Loss:", loss)

print("Accuracy:", accuracy)

A screenshot of a computer

Description automatically generated

from keras import Sequential

from keras.datasets import mnist

import numpy as np

from keras.layers import Dense

from keras.utils import to\_categorical

(train\_images,train\_labels),(test\_images, test\_labels) = mnist.load\_data()

print(train\_images.shape[1:])

#process the data

#1. convert each image of shape 28\*28 to 784 dimensional which will be fed to the network as a single feature

dimData = np.prod(train\_images.shape[1:])

print(dimData)

train\_data = train\_images.reshape(train\_images.shape[0],dimData)

test\_data = test\_images.reshape(test\_images.shape[0],dimData)

#convert data to float and scale values between 0 and 1

train\_data = train\_data.astype('float')

test\_data = test\_data.astype('float')

#scale data

train\_data /=255.0

test\_data /=255.0

#change the labels frominteger to one-hot encoding. to\_categorical is doing the same thing as LabelEncoder()

train\_labels\_one\_hot = to\_categorical(train\_labels)

test\_labels\_one\_hot = to\_categorical(test\_labels)

#creating network

model = Sequential()

model.add(Dense(512, activation='relu', input\_shape=(dimData,)))

model.add(Dense(512, activation='relu'))

model.add(Dense(10, activation='softmax'))

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

history = model.fit(train\_data, train\_labels\_one\_hot, batch\_size=256, epochs=10, verbose=1,

                   validation\_data=(test\_data, test\_labels\_one\_hot))

A table with numbers and letters

Description automatically generated

import matplotlib.pyplot as plt

# Plot training & validation accuracy values

plt.plot(history.history['accuracy'])

plt.plot(history.history['val\_accuracy'])

plt.title('Model accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

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

plt.show()

# Plot training & validation loss values

plt.plot(history.history['loss'])

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

plt.title('Model loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

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

plt.show()

A graph with blue and orange lines

Description automatically generatedA graph with blue lines and orange lines

Description automatically generatedimport matplotlib.pyplot as plt

# select a random image from test data

image\_index = 1234

img = test\_images[image\_index]

# plot the image

plt.imshow(img, cmap='gray')

# reshape image to 1D vector

img = img.reshape((1, 784))

# normalize pixel values

img = img / 255.0

# predict class of image

result = model.predict(img)

print("Predicted digit:", np.argmax(result))

A screen shot of a number

Description automatically generated

import keras

from keras.datasets import mnist

from keras.models import Sequential

from keras.layers import Dense, Dropout

import matplotlib.pyplot as plt

import numpy as np

# load MNIST dataset

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

# normalize pixel values to range [0, 1]

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

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

# convert class labels to binary class matrices

num\_classes = 10

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

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

# create a list of models to train

models = []

# model with 1 hidden layer and tanh activation

model = Sequential()

model.add(Dense(512, activation='tanh', input\_shape=(784,)))

model.add(Dropout(0.2))

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

models.append(('1 hidden layer with tanh', model))

# model with 1 hidden layer and sigmoid activation

model = Sequential()

model.add(Dense(512, activation='sigmoid', input\_shape=(784,)))

model.add(Dropout(0.2))

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

models.append(('1 hidden layer with sigmoid', model))

# model with 2 hidden layers and tanh activation

model = Sequential()

model.add(Dense(512, activation='tanh', input\_shape=(784,)))

model.add(Dropout(0.2))

model.add(Dense(512, activation='tanh'))

model.add(Dropout(0.2))

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

models.append(('2 hidden layers with tanh', model))

# model with 2 hidden layers and sigmoid activation

model = Sequential()

model.add(Dense(512, activation='sigmoid', input\_shape=(784,)))

model.add(Dropout(0.2))

model.add(Dense(512, activation='sigmoid'))

model.add(Dropout(0.2))

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

models.append(('2 hidden layers with sigmoid', model))

# train each model and plot loss and accuracy curves

for name, model in models:

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

    history = model.fit(x\_train.reshape(-1, 784), y\_train, validation\_data=(x\_test.reshape(-1, 784), y\_test),

                        epochs=20, batch\_size=128, verbose=0)

    # plot loss and accuracy curves

    plt.plot(history.history['loss'], label='train\_loss')

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

    plt.plot(history.history['accuracy'], label='train\_accuracy')

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

    plt.title(name)

    plt.xlabel('Epoch')

    plt.legend()

    plt.show()

    # evaluate the model on test data

    loss, accuracy = model.evaluate(x\_test.reshape(-1, 784), y\_test, verbose=0)

    print('{} - Test loss: {:.4f}, Test accuracy: {:.4f}'.format(name, loss, accuracy))

A graph of a graph

Description automatically generated with medium confidence

A graph of a graph with numbers and lines

Description automatically generated with medium confidenceA graph of a graph with text

Description automatically generated with medium confidenceA graph of different layers

Description automatically generated

import keras

from keras.datasets import mnist

from keras.models import Sequential

from keras.layers import Dense, Dropout

import matplotlib.pyplot as plt

import numpy as np

# load MNIST dataset

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

# convert class labels to binary class matrices

num\_classes = 10

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

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

# create a list of models to train

models = []

# model with 1 hidden layer and tanh activation

model = Sequential()

model.add(Dense(512, activation='tanh', input\_shape=(784,)))

model.add(Dropout(0.2))

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

models.append(('1 hidden layer with tanh', model))

# model with 1 hidden layer and sigmoid activation

model = Sequential()

model.add(Dense(512, activation='sigmoid', input\_shape=(784,)))

model.add(Dropout(0.2))

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

models.append(('1 hidden layer with sigmoid', model))

# model with 2 hidden layers and tanh activation

model = Sequential()

model.add(Dense(512, activation='tanh', input\_shape=(784,)))

model.add(Dropout(0.2))

model.add(Dense(512, activation='tanh'))

model.add(Dropout(0.2))

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

models.append(('2 hidden layers with tanh', model))

# model with 2 hidden layers and sigmoid activation

model = Sequential()

model.add(Dense(512, activation='sigmoid', input\_shape=(784,)))

model.add(Dropout(0.2))

model.add(Dense(512, activation='sigmoid'))

model.add(Dropout(0.2))

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

models.append(('2 hidden layers with sigmoid', model))

# train each model and plot loss and accuracy curves

for name, model in models:

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

    history = model.fit(x\_train.reshape(-1, 784), y\_train, validation\_data=(x\_test.reshape(-1, 784), y\_test),

                        epochs=20, batch\_size=128, verbose=0)

    # plot loss and accuracy curves

    plt.plot(history.history['loss'], label='train\_loss')

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

    plt.plot(history.history['accuracy'], label='train\_accuracy')

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

    plt.title(name)

    plt.xlabel('Epoch')

    plt.legend()

    plt.show()

    # evaluate the model on test data

    loss, accuracy = model.evaluate(x\_test.reshape(-1, 784), y\_test, verbose=0)

    print('{} - Test loss: {:.4f}, Test accuracy: {:.4f}'.format(name, loss, accuracy))

A graph of a number of different colored lines

Description automatically generated

A graph of a number of different colored lines

Description automatically generatedA graph of different colored lines

Description automatically generatedA graph of different colored lines

Description automatically generated