**ASSINGMENT 6**

**Name: varna nemulla**

**Id: 700744920**

1. Use the use case in the class:

a. Add more Dense layers to the existing code and check how the accuracy changes.

2. Change the data source to Breast Cancer dataset \* available in the source code folder and make required changes. Report accuracy of the model.

3. Normalize the data before feeding the data to the model and check how the normalization change your accuracy (code given below). from sklearn.preprocessing import StandardScaler sc = StandardScaler()

Source code:

#read the data

data = pd.read\_csv('sample\_data/diabetes.csv')

path\_to\_csv = 'sample\_data/diabetes.csv'

import keras

import pandas

from keras.models import Sequential

from keras.layers.core import Dense, Activation

# 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(4, 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))

read the data

data = pd.read\_csv('sample\_data/breastcancer.csv')

import keras

import pandas as pd

import numpy as np

from keras.models import Sequential

from keras.layers.core import Dense, Activation

from sklearn.datasets import load\_breast\_cancer

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

# load dataset

cancer\_data = load\_breast\_cancer()

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(cancer\_data.data, cancer\_data.target,

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

np.random.seed(155)

my\_nn = Sequential() # create model

my\_nn.add(Dense(20, input\_dim=30, activation='relu')) # hidden layer 1

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

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

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

initial\_epoch=0)

print(my\_nn.summary())

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

#read the data

data = pd.read\_csv('sample\_data/breastcancer.csv')

path\_to\_csv = 'sample\_data/breastcancer.csv'

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

import keras

import pandas as pd

import numpy as np

from keras.models import Sequential

from keras.layers.core import Dense, Activation

from sklearn.datasets import load\_breast\_cancer

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

# load dataset

cancer\_data = load\_breast\_cancer()

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(cancer\_data.data, cancer\_data.target,

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

np.random.seed(155)

my\_nn = Sequential() # create model

my\_nn.add(Dense(20, input\_dim=30, activation='relu')) # hidden layer 1

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

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

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

initial\_epoch=0)

print(my\_nn.summary())

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

Graphical user interface

Description automatically generated with medium confidence

Graphical user interface, text, application

Description automatically generated

Graphical user interface

Description automatically generated with medium confidence

Graphical user interface, text

Description automatically generated



Graphical user interface, text

Description automatically generated

**Description:**

In this program, I have developed a program where I added more layers to the existing code and the accuracy got changed and in the second part changed the data source to breast cancer dataset using available source code folder and made the required changes. In the third part normalized the data before feeding the data to the model and it has changed the normalization of the accurary

**Question 2:**

In-class programming: Use Image Classification on the handwritten digits data set (mnist)

1. Plot the loss and accuracy for both training data and validation data using the history object in the source code.

2. Plot one of the images in the test data, and then do inferencing to check what is the prediction of the model on that single image.

3. We had used 2 hidden layers and Relu activation. Try to change the number of hidden layer and the activation to tanh or sigmoid and see what happens.

4. Run the same code without scaling the images and check the performance?

Source Code:

import keras

from keras.datasets import mnist

from keras.models import Sequential

from keras.layers import Dense, Dropout

import matplotlib.pyplot as plt

# 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 simple neural network model

model = Sequential()

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

model.add(Dropout(0.2))

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

model.add(Dropout(0.2))

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

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

# train the model and record the training history

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

epochs=20, batch\_size=128)

# plot the training and validation accuracy and loss curves

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)

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

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

plt.title('Model Accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

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

plt.subplot(1, 2, 2)

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

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

plt.title('Model Loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

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

plt.show()

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 simple neural network model

model = Sequential()

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

model.add(Dropout(0.2))

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

model.add(Dropout(0.2))

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

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

# train the model

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

epochs=20, batch\_size=128)

# plot one of the images in the test data

plt.imshow(x\_test[0], cmap='gray')

plt.show()

# make a prediction on the image using the trained model

prediction = model.predict(x\_test[0].reshape(1, -1))

print('Model prediction:', np.argmax(prediction))

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))

**Screenshot of source code and output:**

Text

Description automatically generated

Graphical user interface

Description automatically generated with medium confidence

A picture containing graphical user interface

Description automatically generated

Graphical user interface, text

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, text

Description automatically generated

Graphical user interface, text

Description automatically generated

Graphical user interface

Description automatically generated

Graphical user interface, text

Description automatically generated

Graphical user interface

Description automatically generated

Graphical user interface

Description automatically generated

**Description:**

In the second programforPlot the loss and accuracy for both training data and validation data used the history object in the source code. Plot one of the images in the test data, and then did inferencing to check the prediction of the model on that single image. I had used 2 hidden layers and Relu activation. Tried to change the number of hidden layer and the activation to tanh or sigmoid. Ran the same code without scaling the images aßnd check the performance.

Video link: https://drive.google.com/file/d/1b6BZBkNw9LX\_6ut1W7LIzos8RhwrDonC/view?usp=sharing

GitHub Link: