**Lab-8**

Chandra kiran kopparapu

BL.EN. U4AIE23141

Code:

import numpy as np

import os

import cv2

import matplotlib.pyplot as plt

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

from sklearn.neural\_network import MLPClassifier

from sklearn.metrics import accuracy\_score, classification\_report

# === Dataset Path ===

DATASET\_PATH = "C:\\AIO\\Semster Files\\SEMSTER - 4\\ML\\Lab Work\\ML\_Assignment\_08\_BL.EN.U4AIE23138\\Dataset"

# === Load Dataset ===

def load\_glioma\_dataset():

    data = []

    for fname in os.listdir(DATASET\_PATH):

        img\_path = os.path.join(DATASET\_PATH, fname)

        img = cv2.imread(img\_path, cv2.IMREAD\_GRAYSCALE)

        if img is not None:

            img = cv2.resize(img, (64, 64)).flatten()

            data.append(img)

    return np.array(data)

# === A1: Core Functions ===

def sigmoid(x): return 1 / (1 + np.exp(-x))

def tanh(x): return np.tanh(x)

def relu(x): return np.maximum(0, x)

def leaky\_relu(x): return np.where(x > 0, x, 0.01 \* x)

def a1\_demo(X\_sample):

    print("A1: Activations on first image vector:")

    print("Sigmoid:", sigmoid(X\_sample[:5]))

    print("Tanh:", tanh(X\_sample[:5]))

    print("ReLU:", relu(X\_sample[:5]))

    print("Leaky ReLU:", leaky\_relu(X\_sample[:5]))

# === A2-A3: Perceptron ===

def train\_perceptron(X, y, weights, lr=0.01, max\_epochs=1000, threshold=0.01):

    errors = []

    for epoch in range(max\_epochs):

        total\_error = 0

        for i in range(len(X)):

            pred = 1 if np.dot(weights, X[i]) >= 0 else 0

            error = y[i] - pred

            total\_error += error \*\* 2

            weights += lr \* error \* X[i]

        errors.append(total\_error)

        if total\_error <= threshold:

            break

    return weights, errors

# === A4: Learning Rate Plot ===

def a4\_learning\_rate\_plot(X, y):

    rates = np.arange(0.01, 0.11, 0.01)

    epochs\_list = []

    for lr in rates:

        \_, errors = train\_perceptron(X, y.copy(), np.random.rand(X.shape[1]), lr)

        epochs\_list.append(len(errors))

    plt.plot(rates, epochs\_list, marker='o')

    plt.title("A4: Learning Rate vs Epochs")

    plt.xlabel("Learning Rate")

    plt.ylabel("Epochs")

    plt.grid(True)

    plt.show()

# === A5: Repeat with XOR-Like Labels ===

def xor\_like\_labels(X):

    return np.logical\_xor(X[:, 0] > 127, X[:, 1] > 127).astype(int)

# === A6: Perceptron with Sigmoid Activation ===

def a6\_sigmoid\_train(X, y, alpha=0.01, max\_epochs=1000):

    w = np.random.randn(X.shape[1])

    for \_ in range(max\_epochs):

        for xi, yi in zip(X, y):

            pred = sigmoid(np.dot(xi, w))

            error = yi - (pred >= 0.5)

            w += alpha \* error \* xi

    return w

# === A7: Pseudo-Inverse ===

def a7\_pseudo\_inverse(X, y):

    print("A7: Pseudo-inverse solution shape:", np.linalg.pinv(X).shape)

    return np.linalg.pinv(X) @ y

# === A8-A9: Backpropagation ===

def a8\_backprop(X, y, alpha=0.01, max\_epochs=500):

    input\_dim, hidden\_dim = X.shape[1], 10

    w1 = np.random.randn(input\_dim, hidden\_dim)

    w2 = np.random.randn(hidden\_dim, 1)

    errors = []

    for \_ in range(max\_epochs):

        h = sigmoid(X @ w1)

        out = sigmoid(h @ w2).flatten()

        err = y - out

        errors.append(np.sum(err \*\* 2))

        if errors[-1] <= 0.01:

            break

        delta\_out = (err \* out \* (1 - out))[:, None]

        delta\_hid = delta\_out @ w2.T \* h \* (1 - h)

        w2 += alpha \* h.T @ delta\_out

        w1 += alpha \* X.T @ delta\_hid

    return errors

# === A10: Multi-output Mapping ===

def a10\_multi\_output(y):

    return np.array([[0, 1] for \_ in y])

# === A11: sklearn MLPClassifier ===

def a11\_mlp\_classifier(X, y):

    model = MLPClassifier(hidden\_layer\_sizes=(64,), max\_iter=500, random\_state=42)

    model.fit(X, y)

    print("A11: Sklearn MLPClassifier Accuracy:", model.score(X, y))

# === A12: Final Evaluation ===

def a12\_final(X, y):

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

    model = MLPClassifier(hidden\_layer\_sizes=(64,), max\_iter=500)

    model.fit(X\_train, y\_train)

    y\_pred = model.predict(X\_test)

    print("A12: Accuracy:", accuracy\_score(y\_test, y\_pred))

    print("Classification Report:\n", classification\_report(y\_test, y\_pred))

# === Run All ===

X = load\_glioma\_dataset()

y = np.zeros(len(X))  # Only one class

a1\_demo(X[0])                            # A1

\_, errors\_a2 = train\_perceptron(X, y, np.random.rand(X.shape[1]))  # A2-A3

a4\_learning\_rate\_plot(X, y)             # A4

xor\_labels = xor\_like\_labels(X)         # A5

train\_perceptron(X, xor\_labels, np.random.rand(X.shape[1]))        # A5 continued

a6\_sigmoid\_train(X, y)                  # A6

a7\_pseudo\_inverse(X, y)                 # A7

backprop\_errors = a8\_backprop(X, y)     # A8-A9

plt.plot(backprop\_errors)

plt.title("A9: Backprop Error Curve")

plt.xlabel("Epoch")

plt.ylabel("SSE")

plt.grid(True)

plt.show()

print("A10:", a10\_multi\_output(y)[:3])  # A10

a11\_mlp\_classifier(X, y)                # A11

a12\_final(X, y)                         # A12