TASK#03

import numpy as np

import gzip

import matplotlib.pyplot as plt

from urllib import request

from sklearn.preprocessing import OneHotEncoder

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

# Load MNIST data

def load\_mnist\_data():

url\_base = 'http://yann.lecun.com/exdb/mnist/'

files = {

'train\_images': 'train-images-idx3-ubyte.gz',

'train\_labels': 'train-labels-idx1-ubyte.gz',

'test\_images': 't10k-images-idx3-ubyte.gz',

'test\_labels': 't10k-labels-idx1-ubyte.gz',

}

def download\_and\_extract(url, filename):

print(f"Downloading {url}{filename}")

response = request.urlopen(f"{url}{filename}")

with open(filename, 'wb') as f:

f.write(response.read())

with gzip.open(filename, 'rb') as f:

if 'images' in filename:

return np.frombuffer(f.read(), np.uint8, offset=16).reshape(-1, 28 \* 28)

elif 'labels' in filename:

return np.frombuffer(f.read(), np.uint8, offset=8)

data = {}

for key, value in files.items():

data[key] = download\_and\_extract(url\_base, value)

return data['train\_images'], data['train\_labels'], data['test\_images'], data['test\_labels']

train\_images, train\_labels, test\_images, test\_labels = load\_mnist\_data()

# Normalize pixel values

train\_images = train\_images / 255.0

test\_images = test\_images / 255.0

# One-hot encode labels

encoder = OneHotEncoder(sparse=False, categories='auto')

train\_labels = encoder.fit\_transform(train\_labels.reshape(-1, 1))

test\_labels = encoder.transform(test\_labels.reshape(-1, 1))

# Split training data into training and validation sets

X\_train, X\_val, y\_train, y\_val = train\_test\_split(train\_images, train\_labels, test\_size=0.1, random\_state=42)

# Define neural network

class NeuralNetwork:

def \_\_init\_\_(self, input\_size, hidden\_size, output\_size):

self.input\_size = input\_size

self.hidden\_size = hidden\_size

self.output\_size = output\_size

# Initialize weights and biases

self.W1 = np.random.randn(input\_size, hidden\_size)

self.b1 = np.zeros((1, hidden\_size))

self.W2 = np.random.randn(hidden\_size, output\_size)

self.b2 = np.zeros((1, output\_size))

def sigmoid(self, z):

return 1 / (1 + np.exp(-z))

def sigmoid\_derivative(self, z):

return z \* (1 - z)

def softmax(self, z):

exp\_z = np.exp(z - np.max(z, axis=1, keepdims=True))

return exp\_z / np.sum(exp\_z, axis=1, keepdims=True)

def forward(self, X):

self.z1 = np.dot(X, self.W1) + self.b1

self.a1 = self.sigmoid(self.z1)

self.z2 = np.dot(self.a1, self.W2) + self.b2

self.a2 = self.softmax(self.z2)

return self.a2

def backward(self, X, y, output, learning\_rate):

m = y.shape[0]

d\_z2 = output - y

d\_W2 = np.dot(self.a1.T, d\_z2) / m

d\_b2 = np.sum(d\_z2, axis=0, keepdims=True) / m

d\_a1 = np.dot(d\_z2, self.W2.T)

d\_z1 = d\_a1 \* self.sigmoid\_derivative(self.a1)

d\_W1 = np.dot(X.T, d\_z1) / m

d\_b1 = np.sum(d\_z1, axis=0, keepdims=True) / m

# Update weights and biases

self.W1 -= learning\_rate \* d\_W1

self.b1 -= learning\_rate \* d\_b1

self.W2 -= learning\_rate \* d\_W2

self.b2 -= learning\_rate \* d\_b2

def train(self, X, y, epochs, learning\_rate):

for epoch in range(epochs):

output = self.forward(X)

self.backward(X, y, output, learning\_rate)

if (epoch + 1) % 100 == 0:

loss = self.compute\_loss(y, output)

print(f"Epoch {epoch + 1}/{epochs}, Loss: {loss:.4f}")

def compute\_loss(self, y, output):

m = y.shape[0]

log\_likelihood = -np.log(output[range(m), np.argmax(y, axis=1)])

loss = np.sum(log\_likelihood) / m

return loss

def predict(self, X):

output = self.forward(X)

return np.argmax(output, axis=1)

# Initialize and train the neural network

input\_size = 28 \* 28

hidden\_size = 64

output\_size = 10

epochs = 1000

learning\_rate = 0.1

nn = NeuralNetwork(input\_size, hidden\_size, output\_size)

nn.train(X\_train, y\_train, epochs, learning\_rate)

# Evaluate the neural network

def evaluate(model, X, y):

predictions = model.predict(X)

labels = np.argmax(y, axis=1)

accuracy = np.mean(predictions == labels)

return accuracy

train\_accuracy = evaluate(nn, X\_train, y\_train)

val\_accuracy = evaluate(nn, X\_val, y\_val)

test\_accuracy = evaluate(nn, test\_images, test\_labels)

print(f"Training Accuracy: {train\_accuracy:.4f}")

print(f"Validation Accuracy: {val\_accuracy:.4f}")

print(f"Test Accuracy: {test\_accuracy:.4f}")

# Plot a few predictions

def plot\_predictions(model, X, y, num\_images=5):

predictions = model.predict(X)

labels = np.argmax(y, axis=1)

for i in range(num\_images):

plt.imshow(X[i].reshape(28, 28), cmap='gray')

plt.title(f"True Label: {labels[i]}, Predicted: {predictions[i]}")

plt.show()

plot\_predictions(nn, test\_images, test\_labels)