import numpy as np

# Assuming we have loaded your dataset into X and Y variables

# X is the matrix of pixel values and Y is the corresponding labels

# Splitting the dataset

X\_dev = X[:1000]

Y\_dev = Y[:1000]

X\_train = X[1000:]

Y\_train = Y[1000:]

# Normalizing the pixel values

X\_dev = X\_dev / 255.0

X\_train = X\_train / 255.0

# Part-1

## Question-1:

import numpy as np

def init\_params():

np.random.seed(1) # For reproducibility

W1 = np.random.rand(784, 120)

b1 = np.random.rand(120)

W2 = np.random.rand(120, 45)

b2 = np.random.rand(45)

W3 = np.random.rand(45, 10)

b3 = np.random.rand(10)

return {'W1': W1, 'b1': b1, 'W2': W2, 'b2': b2, 'W3': W3, 'b3': b3}

## Question-2:

def ReLU(Z):

return np.maximum(0, Z)

def Softmax(Z):

exp\_scores = np.exp(Z)

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

## Question-3:

def forward\_propagation(X, params):

W1, b1, W2, b2, W3, b3 = params

Z1 = np.dot(X, W1) + b1

A1 = ReLU(Z1)

Z2 = np.dot(A1, W2) + b2

A2 = ReLU(Z2)

Z3 = np.dot(A2, W3) + b3

A3 = Softmax(Z3)

return {'Z1': Z1, 'A1': A1, 'Z2': Z2, 'A2': A2, 'Z3': Z3, 'A3': A3}

## Question-4:

def one\_hot(Y, num\_classes):

num\_samples = len(Y)

encoded = np.zeros((num\_classes, num\_samples))

encoded[Y, np.arange(num\_samples)] = 1

return encoded.T

# Part-2

## Question-1:

def backward\_propagation(X, Y, params, cache):

m = X.shape[0]

A1, A2, A3 = cache['A1'], cache['A2'], cache['A3']

Z1, Z2, Z3 = cache['Z1'], cache['Z2'], cache['Z3']

W1, W2, W3 = params['W1'], params['W2'], params['W3']

dZ3 = A3 - one\_hot(Y, 10)

dW3 = (1 / m) \* np.dot(A2.T, dZ3)

db3 = (1 / m) \* np.sum(dZ3, axis=0)

dA2 = np.dot(dZ3, W3.T)

dZ2 = np.multiply(dA2, np.int64(A2 > 0))

dW2 = (1 / m) \* np.dot(A1.T, dZ2)

db2 = (1 / m) \* np.sum(dZ2, axis=0)

dA1 = np.dot(dZ2, W2.T)

dZ1 = np.multiply(dA1, np.int64(A1 > 0))

dW1 = (1 / m) \* np.dot(X.T, dZ1)

db1 = (1 / m) \* np.sum(dZ1, axis=0)

return {'dW1': dW1, 'db1': db1, 'dW2': dW2, 'db2': db2, 'dW3': dW3, 'db3': db3}

## Question-2:

def update\_params(params, grads, alpha):

W1, b1, W2, b2, W3, b3 = params

dW1, db1, dW2, db2, dW3, db3 = grads

W1 -= alpha \* dW1

b1 -= alpha \* db1

W2 -= alpha \* dW2

b2 -= alpha \* db2

W3 -= alpha \* dW3

b3 -= alpha \* db3

return W1, b1, W2, b2, W3, b3

## Question-3:

def get\_prediction(A3):

return np.argmax(A3, axis=1)

def get\_accuracy(predictions, Y):

return np.mean(predictions == Y) \* 100

## Question-4:

def gradient\_descent(X\_train, Y\_train, alpha=0.1, num\_iterations=1000):

params = init\_params()

for i in range(num\_iterations):

cache = forward\_propagation(X\_train, params)

grads = backward\_propagation(X\_train, Y\_train, params, cache)

params = update\_params(params, grads, alpha)

if i % 10 == 0:

predictions = get\_prediction(cache['A3'])

accuracy = get\_accuracy(predictions, Y\_train)

print(f"Iteration: {i}, Accuracy: {accuracy}%")

return params

# Part-3

## Question-1:

def make\_predictions(X, params):

cache = forward\_propagation(X, params)

predictions = get\_prediction(cache['A3'])

return predictions

## Question-2:

import matplotlib.pyplot as plt

def test\_prediction(index, params):

X\_test = X\_train[index]

Y\_test = Y\_train[index]

prediction = make\_predictions(X\_test, params)

plt.imshow(X\_test.reshape(28, 28), cmap='gray')

plt.axis('off')

plt.title(f"Prediction: {prediction}, True Label: {Y\_test}")

plt.show()