EEE 443

Mini Project 1

Handwritten Digit Recognition by a Multilayer Neural Network

Name:Ahmet Bera Özbolat

ID:21902777

Introduction

For this short project activity, we had to construct and train a neural network using the MNIST dataset in order to enable it to identify handwritten digits from a picture containing 28 by 28 pixels. The network was selected to have two layers: an output layer and a hidden layer with three distinct neuron counts. Another requirement was that three alternative learning coefficients be used in the training process, and that the concepts of gradient descent and backpropagation be applied. We were also required to apply the aforementioned attributes for two distinct scenarios. The tanh activation function was to be used for both layers in one scenario, and the RELU function for the first layer and the sigmoid for the second layer were to be used in the other. Consequently, eighteen distinct neural networks with various attributes were identified. It was asked that we evaluate each's performance in comparison and offer our thoughts on the outcomes.

Implementation

First, there was a need to figure out how to retrieve the data from the database so that the network could be trained. So, using the "MNIST" module, I imported the data into the Python interpreter using the code below after downloading the pertinent dataset from the website supplied in the project instructions.

```
# Step 1: Load image and label data
def load_mnist_data():

    # Initialize the MNIST object
    mndata = MNIST('')
    mndata.gz = True

    # Load training images and labels
    images, labels = mndata.load_training()

    # Load test images and labels
    test_images, test_labels = mndata.load_testing()

    return np.array(images), np.array(labels), np.array(test_images), np.array(test_labels)
```

Figure 1:Mnist Data load

```
Train_input,Train_label,Test_input,Test_label= load_mnist_data()
```

Figure 2: Data gathering

After implementing the data gathering function i implemented the weight initialization code according to Project manual

```
def initialize_weights(hidden_size):
    weights_hidden = np.random.uniform(-0.01, 0.01, size=(hidden_size, 784))
    weights_output = np.random.uniform(-0.01, 0.01, size=(10, hidden_size))

    return weights_hidden, weights_output
```

Figure 3: Weight initialize

For the forward propagation part i used to logic we have discussed in class which is,

$$Z_1=W_1^t \cdot x$$

$$o_1=\text{activation}(v_1)$$

$$Z_2=W_2^t \cdot o_1$$

$$O_2=\text{activation}(Z_2)$$

```
def forward_pass(inputs, weights_input_hidden, weights_hidden_output, activation_hidden, activation_output):
    hidden_inputs = np.dot(weights_input_hidden, inputs)
    hidden_outputs = activation(hidden_inputs, activation_hidden)
    final_inputs = np.dot(weights_hidden_output, hidden_outputs)
    final_outputs = activation(final_inputs, activation_output)
    return hidden_outputs, final_outputs
```

Figure 4: Forward propagation

For Case 1, I have used the tanh activation function, and for Case 2, relu activation for the hidden layer, and sigmoid activation for the output layer.

```python
def tanh_activation(x):
    return np.tanh(x)
```

```python
def relu_activation(x):
    return np.maximum(x, 0)
```

```python
def sigmoid_activation(x):
    y = expit(x)
    return y
```

Figure 5: Activation Functions

For preassigning the output neurons to digits i have used basic if else function, Thus desired output are can be assigned using this function.

```python
def desired_output_tanh(label,polarity):
    if polarity == -1:
        d = -np.ones((10,1))
        d[label] = 1
        return d
    else:
        d = np.zeros((10,1))
        d[label] = 1
        return d
```

Figure 6: Assigning digits to the neurons

For back propabagation algorithm i used the approach that we discussed in the class

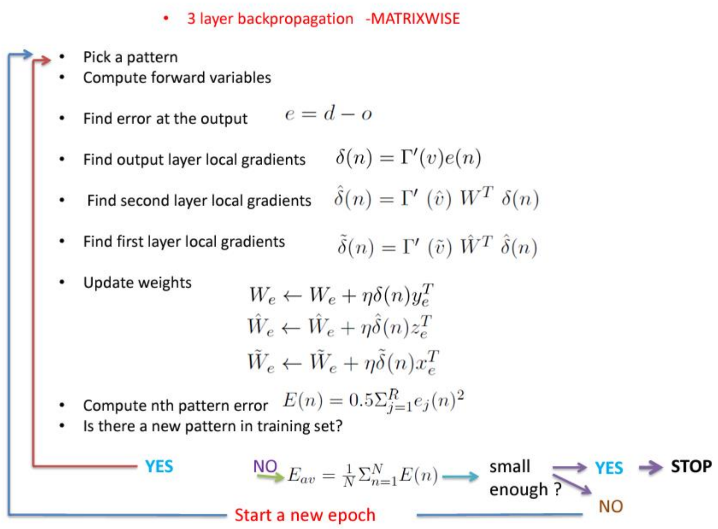

Figure 7: Backpropagation Algorithm from lecture notes

To find local gradients, I first implement a function that takes values output and desired output and calculates and returns the error.

```
def find_error(output,desired):
    error = desired - output
    return error
```

Figure 8: Error calculation

For the First case, using the derivative of tanh activation ($1/2-1/2*o^2$) and the final output of the network and algorithm mention above we can find the output local gradients.

```
def output_gradient(error,final_output):
    derivative=np.eye(10)*(1/2)*(1-final_output*final_output)
    gradient=np.dot(derivative,error)
    output_gradient=np.reshape(gradient,(10,1))
    return output_gradient
```

Figure 9: Output Local gradient calculation

Just like before using the derivative of tanh and the algorithm, in this case we need to use the output gradient and output of the first(hidden) layer.

```
def input_gradient(layer_num,first_output,output_gradient,weight_output):
    derivative=np.eye(layer_num)*(1/2)*(1-first_output*first_output)
    gradient=np.dot(np.dot(derivative,np.transpose(weight_output)),output_gradient)
    input_gradient=np.reshape(gradient,(layer_num,1))
    return input_gradient
```

Figure 10: Hidden layer local gradients calculation

For Case 2, since the activation functions are different I need to modify the functions above, because derivatives are different and also in this case we have a unipolar case.

```
def output_gradient2(error,final_output):
    derivative=np.eye(10)*(final_output-final_output*final_output)
    gradient=np.dot(derivative,error)
    output_gradient=np.reshape(gradient,(10,1))
    return output_gradient
```

Figure 11:Output Local gradient calculation for Case 2

```python
def input_gradient2(layer_num,first_output,output_gradient,weight_output):
    derivative=np.eye(layer_num)*relu_derivative(first_output)
    gradient=np.dot(np.dot(derivative,np.transpose(weight_output)),output_gradient)
    input_gradient=np.reshape(gradient,(layer_num,1))
    return input_gradient
```

Figure 12: Hidden layer local gradients calculation for Case 2

```python
def relu_derivative(x):
    x[x>=0]=1
    x[x<0]=0
    return x
```

Figure 13:Derivative Relu Function

Using the functions above i have implemented a back propagation function so that in every epoch we can update the weights and find the error.

```python
def backpropagation_tanh(inputs, labels, weights_hidden_output, weights_input_hidden, activation_hidden, activation_output,
                         learning_rate, layer_num):
    hidden_outputs, final_outputs = forward_pass(inputs, weights_input_hidden, weights_hidden_output, activation_hidden,
                                                 activation_output)
    desired = desired_output_tanh(labels, -1)
    error = find_error(final_outputs, desired)
    output_gradients = output_gradient2(error, final_outputs)
    weights_hidden_output_updated = weights_hidden_output + learning_rate * np.dot(output_gradients, hidden_outputs.T)
    input_gradients = input_gradient(layer_num, hidden_outputs, output_gradients, weights_hidden_output)
    weights_input_hidden_updated = weights_input_hidden + learning_rate * np.dot(input_gradients, inputs.T)

    return error, weights_hidden_output_updated, weights_input_hidden_updated
```

Figure 14: Backpropagation for Case 1

```python
def backpropagation_sigmoid(inputs, labels, weights_hidden_output, weights_input_hidden,
                            activation_hidden, activation_output, learning_rate, layer_num):
    hidden_outputs, final_outputs = forward_pass(inputs, weights_input_hidden,
                                                 weights_hidden_output, activation_hidden, activation_output)
    desired = desired_output_tanh(labels, 0)
    error = find_error(final_outputs, desired)
    output_gradients = output_gradient2(error, final_outputs)
    weights_hidden_output_updated = weights_hidden_output + learning_rate * np.dot(output_gradients, hidden_outputs.T)
    input_gradients = input_gradient2(layer_num, hidden_outputs, output_gradients, weights_hidden_output)
    weights_input_hidden_updated = weights_input_hidden + learning_rate * np.dot(input_gradients, inputs.T)

    return error, weights_hidden_output_updated, weights_input_hidden_updated
```

Figure 15:: Backpropagation for Case 2

```python
def train(inputs, labels,test_images ,test_label,weights_hidden_output,
          weights_input_hidden, activation_hidden, activation_output, learning_rate, layer_num, batch_size, epochs):
    Egitim_error = 0.0
    startTime = time.time()

    for epoch in range(epochs):
        Error_Toplam = 0.0

        for i in range(batch_size):
            flattened_data = np.reshape(inputs[i], (784, 1)) / 255
            error, weights_hidden_output, weights_input_hidden = backpropagation_tanh(flattened_data, labels[i],
                                                                                        weights_hidden_output, weights_input_hi
            Error_Toplam += np.sum(error**2 * 0.5)

        Epoch_mean = Error_Toplam / batch_size
        Egitim_error += Epoch_mean

        print("Epoch Completed: ", epoch + 1, "\nMean Squares Error of Epoch: ", Epoch_mean)

    Egitim_errorMean = Egitim_error / epochs
    stopTime = time.time()
    print("Training Completed \n CPU Time: ", stopTime - startTime, " seconds", "\n Overall Error:", Egitim_errorMean)

    Error_Toplam_2 = 0.0
    numOfMissClassification = 0

    for i in range(10000):
        testData = np.reshape(test_images[i], (784, 1)) / 255
        layer_output,final_output=forward_pass(testData, weights_input_hidden, weights_hidden_output,
                                               activation_hidden, activation_output)
        desired=desired_output_tanh(test_label[i],-1)
        error_2=find_error(final_output,desired)
        Error_Toplam_2 += np.sum(error_2**2 * 0.5)

        predicted = np.argmax(final_output)

        if test_label[i] != predicted:
            numOfMissClassification += 1

    misclas_per = numOfMissClassification / 10000 * 100

    testSetErrorMean = Error_Toplam_2 / 10000

    print("Number Of Misclasification:",numOfMissClassification,"\nError Percentage: %",misclas_per,
          "\nTest Data Mean Square Error:",testSetErrorMean)
```

Figure 16: Training for case 1

```python
def train2(inputs, labels,test_images ,test_label,weights_hidden_output, weights_input_hidden,
           activation_hidden, activation_output, learning_rate, layer_num, batch_size, epochs):
    Egitim_error = 0.0
    startTime = time.time()

    for epoch in range(epochs):
        Error_Toplam = 0.0

        for i in range(batch_size):
            flattened_data = np.reshape(inputs[i], (784, 1)) / 255
            error, weights_hidden_output, weights_input_hidden = backpropagation_sigmoid(flattened_data, labels[i],
                                                                                           weights_hidden_output, weights_input
            Error_Toplam += np.sum(error**2 * 0.5)

        Epoch_mean = Error_Toplam / batch_size
        Egitim_error += Epoch_mean

        print("Epoch Completed: ", epoch + 1, "\nMean Squares Error of Epoch: ", Epoch_mean)

    Egitim_errorMean = Egitim_error / epochs
    stopTime = time.time()
    print("Training Completed \n CPU Time: ", stopTime - startTime, " seconds", "\n Overall Error:", Egitim_errorMean)

    Error_Toplam_2 = 0.0
    numOfMissClassification = 0

    for i in range(10000):
        testData = np.reshape(test_images[i], (784, 1)) / 255
        layer_output,final_output=forward_pass(testData, weights_input_hidden, weights_hidden_output,
                                               activation_hidden, activation_output)
        desired=desired_output_tanh(test_label[i],0)
        error_2=find_error(final_output,desired)
        Error_Toplam_2 += np.sum(error_2**2 * 0.5)

        predicted = np.argmax(final_output)

        if test_label[i] != predicted:
            numOfMissClassification += 1

    misclas_per = numOfMissClassification / 10000 * 100

    testSetErrorMean = Error_Toplam_2 / 10000

    print("Number Of Misclasification:",numOfMissClassification,"\nError Percentage: %",misclas_per,
          "\nTest Data Mean Square Error:",testSetErrorMean)
```

Figure 17: Training for Case 2

```
CPU Time:  146.94421076774597  seconds
Overall Error: 0.12744526343148282
Number Of Misclasification: 1243
Error Percentage: % 12.43
Test Data Mean Square Error: 0.10736092332576916
```

Figure 18: Finished Training Example

Case 1	Epoch	Training MSE	Test MSE	Misclassifaction Count	Error	Time
N = 300 η = 0.01	50	0.25806275776263987	0.4759017200596294	1276	% 12.76	161.4821424484253 seconds
N = 300 η = 0.05	50	0.1774294174864672	0.4271915197828303	1162	% 11.62	147.22417664527893 seconds
N = 300 η = 0.09	50	0.18325211035133193	0.4313085822923223	1174	% 11.74	162.99382138252258 seconds
N = 500 η = 0.01	50	0.25105003088518474	0.5013361992836625	1340	% 13.4	341.2212908267975 seconds
N = 500 η = 0.05	50	0.21441544562591347	0.4639175036412674	1227	% 12.27	308.33640909194946 seconds
N = 500 η = 0.09	50	0.36082319154956094	0.455684921233564	1142	% 11.42	312.27305579185486 seconds
N = 1000 η = 0.01	50	0.24067863504783912	0.51612849210301	1365	% 13.65	939.6943418979645 seconds
N = 1000 η = 0.05	50	0.26688143577430684	0.5656537493651452	1477	% 14.77	948.1471726894379 seconds
N = 1000 η = 0.09	50	0.6350243178053823	0.6534311881593174	1505	% 15.049	953.5774519443512 seconds

Table 1: Case 1 training results

Best Case Tanh(Full Data)	Epoch	Training MSE	Test MSE	Misclassifaction Count	Error	Time
N = 300 η = 0.01	50	0.053254621245328454	0.07958338256383457	204	% 2.04	6975.323815584183 seconds

N = 300 η = 0.05	50	0.0956091841760674	0.10671934750944176	246	% 2.46	6993.787258863449 seconds
N = 300 η = 0.09	50	0.21015199050574943	0.18950770235084027	452	% 4.52	7036.762281179428 seconds

Table 2: Case 1 full data training results

Case 2	Epoch	Training MSE	Test MSE	Misclassification Count	Error	Time
N = 300 η = 0.01	50	0.12744526343148282	0.10736092332576916	1243	% 12.43	146.94421076774597 seconds
N = 300 η = 0.05	50	0.044155460608273536	0.11475729012293132	1340	% 13.4	137.5495674610138 seconds
N = 300 η = 0.09	50	0.03570633531710205	0.1150534015227812	1339	% 13.389	138.76358938217163 seconds
N = 500 η = 0.01	50	0.1223620287414758	0.1075738447440214	1234	% 13.4	287.11202931404114 seconds
N = 500 η = 0.05	50	0.042879823397784386	0.11589868847146527	1340	% 13.4	289.66985964775085 seconds
N = 500 η = 0.09	50	0.034445117092422886	0.11805976997627661	1349	% 13.489	307.20567870140076 seconds
N = 1000 η = 0.01	50	0.11292003663978464	0.10668177079003435	1242	% 12.42	1010.814615726471 seconds
N = 1000 η = 0.05	50	0.0402508954875381	0.11584552784618425	1348	% 13.48	943.7588999271393 seconds
N = 1000 η = 0.09	50	0.0328981000991999	0.11622277586171267	1318	% 13.18	943.9221329689026 seconds

Table 3: Case 2 Training results

Best Case sigmoid(Full Data)	Epoch	Training MSE	Test MSE	Misclassifaction Count	Error	Time

N = 300 η = 0.01	50	0.05400413424826864 4	0.0624026299828359 6	707	% 7.07	7537.953191757202 seconds

Table 4: Case 2 Full data training

Unforrunately i did not have time to train case for whole data but it can be clearly seen from table 1 and table 3 best case for the digit recognition is case 2. Because it gives less error and take less time so i have train the code with mini batch sizes.

Best Case sigmoid(mini batch)	Epoch	Training MSE	Test MSE	Misclassifaction Count	Error	Time
N=10	50	0.5940877374704915	0.482637744894622	8629	% 86.29	6.442271709442139 seconds
N = 50	50	0.47201739541516124	0.434081906687013 3	6620	% 66.2	32.27644324302673 seconds
N = 100	50	0.3592662317089257	0.296509591008686 35	4285	% 42.85	70.8335976600647 s econds

Table 5: CASE 2 mini batch results

For the weight regulization mentioned in lab manual in the last part I have modified my Backpropagation function for case 2. According the equation below,

$$W \leftarrow (1 - \lambda\,\eta W) - \eta \frac{\partial E}{\partial W}$$

```
def backpropagation_sigmoid2(inputs, labels, weights_hidden_output, weights_input_hidden,
                             activation_hidden, activation_output, learning_rate, layer_num,lambda_coef):
    hidden_outputs, final_outputs = forward_pass(inputs, weights_input_hidden, weights_hidden_output,
                                                 activation_hidden, activation_output)
    desired = desired_output_tanh(labels, 0)
    error = find_error(final_outputs, desired)
    output_gradients = output_gradient2(error, final_outputs)
    weights_hidden_output_updated = (1-learning_rate*lambda_coef)*weights_hidden_output +
    learning_rate * np.dot(output_gradients,hidden_outputs.T)
    input_gradients = input_gradient2(layer_num, hidden_outputs, output_gradients, weights_hidden_output)
    weights_input_hidden_updated = (1-learning_rate*lambda_coef)*weights_input_hidden +
    learning_rate * np.dot(input_gradients, inputs.T)

    return error, weights_hidden_output_updated, weights_input_hidden_updated
```

Figure 19: Modified Case Backpropagation

Case 2 (L2 weight regulization)	Epoch	Training MSE	Test MSE	Misclassifaction Count	Error	Time
Λ=0.01	50	0.3880576842175039	0.3215251564459645	4419	% 44.1900000000000 05	103.54056930541992 seconds

Λ=0.001	50	0.36457905282035413	0.299082173635 4691	4253	% 42.53	105.4805541038 5132 seconds

When these results are compared to the data acquired while determining the best performing mini batch size, the accuracy is reduced. Given that the L2 regularization process is used to address the overfitting issue, this is an expected result. This was not the case because the batch size is not large. Despite the lack of over-training, the accuracy was decreased when the network was penalized for raising its weights. When the network is trained using all of the training data once more, the impact of the L2 regularization becomes more apparent. This configuration would show whether this process is effective in addressing the overfitting issue. To verify this, I used L2 regularization to train a second case network, which I then compared to my results in Table 4. The L2 regularization has significantly increased the accuracy of the network with the test dataset, as can be seen when comparing the results above with those in Table 4. What led to the rise in error rates is also clarified by this.

Conclusion

Two different examples were thoroughly investigated in this in-depth investigation of developing a neural network for handwritten digit recognition using the MNIST dataset. A thorough assessment was carried out using a range of activation functions, learning rates, and regularization strategies. The results show that Case 2, which had a different activation function design, performed better in terms of accuracy and training duration. Notably, the addition of L2 regularization eventually contributed to improving the network's performance, although initially resulting in a little drop in accuracy. This finding emphasizes how important the regularization technique is in mitigating overfitting issues and significantly increasing accuracy when training the network using the full dataset. In the end, this research highlights how important regularization strategies and activation functions are in determining how accurate and resilient neural networks are, especially when dealing with challenging recognition tasks like handwritten digit identification

APPENDICES

CODE

APPENDIX A

```
from mnist import MNIST

import numpy as np

from scipy.special import expit

import time


# Step 1: Load image and label data

def load_mnist_data():


    # Initialize the MNIST object

    mndata = MNIST('')

    mndata.gz = True


    # Load training images and labels

    images, labels = mndata.load_training()


    # Load test images and labels

    test_images, test_labels = mndata.load_testing()


    return np.array(images), np.array(labels), np.array(test_images), np.array(test_labels)




def initialize_weights(hidden_size):

    weights_hidden = np.random.uniform(-0.01, 0.01, size=(hidden_size, 784))

    weights_output = np.random.uniform(-0.01, 0.01, size=(10, hidden_size))


    return weights_hidden, weights_output
```

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

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

def sigmoid_activation(x):
    y = expit(x)
    return y

    def desired_output_tanh(label,notDesiredValue):
        if notDesiredValue == -1:
            d = -np.ones((10,1))
            d[label] = 1
            return d
        else:
            d = np.zeros((10,1))
            d[label] = 1
            return d

def desired_output_sigmoid(label):
    desired = np.eye(10)[label]

    return desired

def find_error(output,desired):
    error = desired - output
    return error

def activation(x, ActivationFunction):
```

```
        return ActivationFunction(x)


def forward_pass(inputs, weights_input_hidden, weights_hidden_output, activation_hidden,
activation_output):
    hidden_inputs = np.dot(weights_input_hidden, inputs)
    hidden_outputs = activation(hidden_inputs, activation_hidden)
    final_inputs = np.dot(weights_hidden_output, hidden_outputs)
    final_outputs = activation(final_inputs, activation_output)
    return hidden_outputs, final_outputs


def output_gradient(error,final_output):
    derivative=np.eye(10)*(1/2)*(1-final_output*final_output)
    gradient=np.dot(derivative,error)
    output_gradient=np.reshape(gradient,(10,1))
    return output_gradient


def output_gradient2(error,final_output):
    derivative=np.eye(10)*(final_output-final_output*final_output)
    gradient=np.dot(derivative,error)
    output_gradient=np.reshape(gradient,(10,1))
    return output_gradient


def input_gradient(layer_num,first_output,output_gradient,weight_output):
    derivative=np.eye(layer_num)*(1/2)*(1-first_output*first_output)
    gradient=np.dot(np.dot(derivative,np.transpose(weight_output)),output_gradient)
    input_gradient=np.reshape(gradient,(layer_num,1))
    return input_gradient
```

```
def relu_derivative(x):
    x[x>=0]=1
    x[x<0]=0
    return x


def input_gradient2(layer_num,first_output,output_gradient,weight_output):
    derivative=np.eye(layer_num)*relu_derivative(first_output)
    gradient=np.dot(np.dot(derivative,np.transpose(weight_output)),output_gradient)
    input_gradient=np.reshape(gradient,(layer_num,1))
    return input_gradient




def backpropagation_tanh(inputs, labels, weights_hidden_output, weights_input_hidden, 
activation_hidden, activation_output, learning_rate, layer_num):
    hidden_outputs, final_outputs = forward_pass(inputs, weights_input_hidden, 
weights_hidden_output, activation_hidden, activation_output)
    desired = desired_output_tanh(labels, -1)
    error = find_error(final_outputs, desired)
    output_gradients = output_gradient2(error, final_outputs)
    weights_hidden_output_updated = weights_hidden_output + learning_rate * 
np.dot(output_gradients, hidden_outputs.T)
    input_gradients = input_gradient(layer_num, hidden_outputs, output_gradients, 
weights_hidden_output)
    weights_input_hidden_updated = weights_input_hidden + learning_rate * np.dot(input_gradients, 
inputs.T)

    return error, weights_hidden_output_updated, weights_input_hidden_updated



def backpropagation_tanh2(inputs, labels, weights_hidden_output, weights_input_hidden, 
activation_hidden, activation_output, learning_rate, layer_num,lambda_coef):
    hidden_outputs, final_outputs = forward_pass(inputs, weights_input_hidden, 
weights_hidden_output, activation_hidden, activation_output)
```

```
    desired = desired_output_tanh(labels, -1)
    error = find_error(final_outputs, desired)
    output_gradients = output_gradient2(error, final_outputs)
    weights_hidden_output_updated = (1-learning_rate*lambda_coef)*weights_hidden_output + learning_rate * np.dot(output_gradients, hidden_outputs.T)
    input_gradients = input_gradient(layer_num, hidden_outputs, output_gradients, weights_hidden_output)
    weights_input_hidden_updated = (1-learning_rate*lambda_coef)*weights_input_hidden + learning_rate * np.dot(input_gradients, inputs.T)

    return error, weights_hidden_output_updated, weights_input_hidden_updated

def backpropagation_sigmoid(inputs, labels, weights_hidden_output, weights_input_hidden, activation_hidden, activation_output, learning_rate, layer_num):
    hidden_outputs, final_outputs = forward_pass(inputs, weights_input_hidden, weights_hidden_output, activation_hidden, activation_output)
    desired = desired_output_tanh(labels, 0)
    error = find_error(final_outputs, desired)
    output_gradients = output_gradient2(error, final_outputs)
    weights_hidden_output_updated = weights_hidden_output + learning_rate * np.dot(output_gradients, hidden_outputs.T)
    input_gradients = input_gradient2(layer_num, hidden_outputs, output_gradients, weights_hidden_output)
    weights_input_hidden_updated = weights_input_hidden + learning_rate * np.dot(input_gradients, inputs.T)

    return error, weights_hidden_output_updated, weights_input_hidden_updated

def backpropagation_sigmoid2(inputs, labels, weights_hidden_output, weights_input_hidden, activation_hidden, activation_output, learning_rate, layer_num,lambda_coef):
    hidden_outputs, final_outputs = forward_pass(inputs, weights_input_hidden, weights_hidden_output, activation_hidden, activation_output)
    desired = desired_output_tanh(labels, 0)
    error = find_error(final_outputs, desired)
```

```
    output_gradients = output_gradient2(error, final_outputs)

    weights_hidden_output_updated = (1-learning_rate*lambda_coef)*weights_hidden_output +
learning_rate * np.dot(output_gradients,hidden_outputs.T)

    input_gradients = input_gradient2(layer_num, hidden_outputs, output_gradients,
weights_hidden_output)

    weights_input_hidden_updated = (1-learning_rate*lambda_coef)*weights_input_hidden +
learning_rate * np.dot(input_gradients, inputs.T)


    return error, weights_hidden_output_updated, weights_input_hidden_updated




def train(inputs, labels,test_images ,test_label,weights_hidden_output, weights_input_hidden,
activation_hidden, activation_output, learning_rate, layer_num, batch_size, epochs):

    trainingError = 0.0

    startTime = time.time()


    for epoch in range(epochs):

        cumulativeError = 0.0


        for i in range(batch_size):

            flattened_data = np.reshape(inputs[i], (784, 1)) / 255

            error, weights_hidden_output, weights_input_hidden =
backpropagation_tanh(flattened_data, labels[i], weights_hidden_output, weights_input_hidden,
activation_hidden, activation_output, learning_rate, layer_num)

            cumulativeError += np.sum(error**2 * 0.5)


        epochErrorMean = cumulativeError / batch_size

        trainingError += epochErrorMean


        print("Epoch Completed: ", epoch + 1, "\nMean Squares Error of Epoch: ", epochErrorMean)


    trainingErrorMean = trainingError / epochs

    stopTime = time.time()
```

```
    print("Training Completed!\nCPU Time: ", stopTime - startTime, " seconds", "\nOverall Error:",
trainingErrorMean)

    cumulativeError_2 = 0.0
    numOfMissClassification = 0

    for i in range(10000):
        testData = np.reshape(test_images[i], (784, 1)) / 255
        layer_output,final_output=forward_pass(testData, weights_input_hidden,
weights_hidden_output,activation_hidden, activation_output)
        desired=desired_output_tanh(test_label[i],-1)
        error_2=find_error(final_output,desired)
        cumulativeError_2 += np.sum(error_2**2 * 0.5)

        predicted = np.argmax(final_output)

        if test_label[i] != predicted:
            numOfMissClassification += 1

    misclasificationPercentage = numOfMissClassification / 10000 * 100

    testSetErrorMean = cumulativeError_2 / 10000

    print("Number Of Misclasification:",numOfMissClassification,"\nError Percentage:
%",misclasificationPercentage,"\nTest Data Mean Square Error:",testSetErrorMean)



Train_input,Train_label,Test_input,Test_label= load_mnist_data()


weights_hidden, weights_output=initialize_weights(300)
```

```
activation_hidden_1= tanh_activation

activation_output_1= tanh_activation


learning_rate=0.01

epochs=50

batch_size=1250

layer_num=300


train(Train_input,Train_label,Test_input ,Test_label,weights_output,
weights_hidden,activation_hidden_1, activation_output_1,
learning_rate,layer_num,batch_size,epochs)


weights_hidden, weights_output=initialize_weights(300)


activation_hidden_1= tanh_activation

activation_output_1= tanh_activation


learning_rate=0.05

epochs=50

batch_size=1250

layer_num=300


train(Train_input,Train_label,Test_input ,Test_label,weights_output,
weights_hidden,activation_hidden_1, activation_output_1,
learning_rate,layer_num,batch_size,epochs)


weights_hidden, weights_output=initialize_weights(300)


learning_rate=0.05

epochs=50

batch_size=1250
```

```
layer_num=300

train(Train_input,Train_label,Test_input ,Test_label,weights_output,
weights_hidden,activation_hidden_1, activation_output_1,
learning_rate,layer_num,batch_size,epochs)


weights_hidden, weights_output=initialize_weights(500)

learning_rate=0.01

epochs=50

batch_size=1250

layer_num=500

train(Train_input,Train_label,Test_input ,Test_label,weights_output,
weights_hidden,activation_hidden_1, activation_output_1,
learning_rate,layer_num,batch_size,epochs)


weights_hidden, weights_output=initialize_weights(500)

learning_rate=0.05

epochs=50

batch_size=1250

layer_num=500

train(Train_input,Train_label,Test_input ,Test_label,weights_output,
weights_hidden,activation_hidden_1, activation_output_1,
learning_rate,layer_num,batch_size,epochs)
```

```
weights_hidden, weights_output=initialize_weights(500)

learning_rate=0.09

epochs=50

batch_size=1250

layer_num=500


train(Train_input,Train_label,Test_input ,Test_label,weights_output,
weights_hidden,activation_hidden_1, activation_output_1,
learning_rate,layer_num,batch_size,epochs)
```

```
weights_hidden, weights_output=initialize_weights(1000)

learning_rate=0.01

epochs=50

batch_size=1250

layer_num=1000


train(Train_input,Train_label,Test_input ,Test_label,weights_output,
weights_hidden,activation_hidden_1, activation_output_1,
learning_rate,layer_num,batch_size,epochs)
```

```
weights_hidden, weights_output=initialize_weights(1000)
```

learning_rate=0.05

epochs=50

batch_size=1250

layer_num=1000

train(Train_input,Train_label,Test_input ,Test_label,weights_output, weights_hidden,activation_hidden_1, activation_output_1, learning_rate,layer_num,batch_size,epochs)

weights_hidden, weights_output=initialize_weights(1000)

learning_rate=0.09

epochs=50

batch_size=1250

layer_num=1000

train(Train_input,Train_label,Test_input ,Test_label,weights_output, weights_hidden,activation_hidden_1, activation_output_1, learning_rate,layer_num,batch_size,epochs)

weights_hidden, weights_output=initialize_weights(300)

learning_rate=0.01

epochs=50

batch_size=60000

layer_num=300

```
train(Train_input,Train_label,Test_input ,Test_label,weights_output,
weights_hidden,activation_hidden_1, activation_output_1,
learning_rate,layer_num,batch_size,epochs)
```

```
weights_hidden, weights_output=initialize_weights(300)

learning_rate=0.05

epochs=50

batch_size=60000

layer_num=300

train(Train_input,Train_label,Test_input ,Test_label,weights_output,
weights_hidden,activation_hidden_1, activation_output_1,
learning_rate,layer_num,batch_size,epochs)
```

```
weights_hidden, weights_output=initialize_weights(300)

learning_rate=0.09

epochs=50

batch_size=60000

layer_num=300

train(Train_input,Train_label,Test_input ,Test_label,weights_output,
weights_hidden,activation_hidden_1, activation_output_1,
learning_rate,layer_num,batch_size,epochs)
```

```
def train2(inputs, labels,test_images ,test_label,weights_hidden_output, weights_input_hidden, activation_hidden, activation_output, learning_rate, layer_num, batch_size, epochs):

    trainingError = 0.0

    startTime = time.time()


    for epoch in range(epochs):

        cumulativeError = 0.0


        for i in range(batch_size):

            flattened_data = np.reshape(inputs[i], (784, 1)) / 255

            error, weights_hidden_output, weights_input_hidden = backpropagation_sigmoid(flattened_data, labels[i], weights_hidden_output, weights_input_hidden, activation_hidden, activation_output, learning_rate, layer_num)

            cumulativeError += np.sum(error**2 * 0.5)


        epochErrorMean = cumulativeError / batch_size

        trainingError += epochErrorMean


        print("Epoch Completed: ", epoch + 1, "\nMean Squares Error of Epoch: ", epochErrorMean)


    trainingErrorMean = trainingError / epochs

    stopTime = time.time()

    print("Training Completed!\nCPU Time: ", stopTime - startTime, " seconds", "\nOverall Error:", trainingErrorMean)


    cumulativeError_2 = 0.0

    numOfMissClassification = 0


    for i in range(10000):

        testData = np.reshape(test_images[i], (784, 1)) / 255

        layer_output,final_output=forward_pass(testData, weights_input_hidden, weights_hidden_output,activation_hidden, activation_output)

        desired=desired_output_tanh(test_label[i],0)
```

```
        error_2=find_error(final_output,desired)
        cumulativeError_2 += np.sum(error_2**2 * 0.5)

        predicted = np.argmax(final_output)

        if test_label[i] != predicted:
            numOfMissClassification += 1

    misclasificationPercentage = numOfMissClassification / 10000 * 100

    testSetErrorMean = cumulativeError_2 / 10000

    print("Number Of Misclasification:",numOfMissClassification,"\nError Percentage:
%",misclasificationPercentage,"\nTest Data Mean Square Error:",testSetErrorMean)

weights_hidden, weights_output=initialize_weights(300)

activation_hidden_1= relu_activation
activation_output_1= sigmoid_activation

weights_hidden, weights_output=initialize_weights(300)
learning_rate=0.01
epochs=50
batch_size=1250
layer_num=300

train2(Train_input,Train_label,Test_input ,Test_label,weights_output,
weights_hidden,activation_hidden_1, activation_output_1,
learning_rate,layer_num,batch_size,epochs)

weights_hidden, weights_output=initialize_weights(300)
learning_rate=0.05
```

```
epochs=50

batch_size=1250

layer_num=300


train2(Train_input,Train_label,Test_input ,Test_label,weights_output,
weights_hidden,activation_hidden_1, activation_output_1,
learning_rate,layer_num,batch_size,epochs)


weights_hidden, weights_output=initialize_weights(300)

learning_rate=0.09

epochs=50

batch_size=1250

layer_num=300


train2(Train_input,Train_label,Test_input ,Test_label,weights_output,
weights_hidden,activation_hidden_1, activation_output_1,
learning_rate,layer_num,batch_size,epochs)


weights_hidden, weights_output=initialize_weights(500)

learning_rate=0.01

epochs=50

batch_size=1250

layer_num=500


train2(Train_input,Train_label,Test_input ,Test_label,weights_output,
weights_hidden,activation_hidden_1, activation_output_1,
learning_rate,layer_num,batch_size,epochs)


weights_hidden, weights_output=initialize_weights(500)

learning_rate=0.05

epochs=50

batch_size=1250

layer_num=500
```

```
train2(Train_input,Train_label,Test_input ,Test_label,weights_output,
weights_hidden,activation_hidden_1, activation_output_1,
learning_rate,layer_num,batch_size,epochs)

weights_hidden, weights_output=initialize_weights(500)

learning_rate=0.09

epochs=50

batch_size=1250

layer_num=500

train2(Train_input,Train_label,Test_input ,Test_label,weights_output,
weights_hidden,activation_hidden_1, activation_output_1,
learning_rate,layer_num,batch_size,epochs)

weights_hidden, weights_output=initialize_weights(1000)

learning_rate=0.01

epochs=50

batch_size=1250

layer_num=1000

train2(Train_input,Train_label,Test_input ,Test_label,weights_output,
weights_hidden,activation_hidden_1, activation_output_1,
learning_rate,layer_num,batch_size,epochs)

weights_hidden, weights_output=initialize_weights(1000)

learning_rate=0.05

epochs=50

batch_size=1250

layer_num=1000

train2(Train_input,Train_label,Test_input ,Test_label,weights_output,
weights_hidden,activation_hidden_1, activation_output_1,
learning_rate,layer_num,batch_size,epochs)
```

```
weights_hidden, weights_output=initialize_weights(1000)

learning_rate=0.09

epochs=50

batch_size=1250

layer_num=1000


train2(Train_input,Train_label,Test_input ,Test_label,weights_output,
weights_hidden,activation_hidden_1, activation_output_1,
learning_rate,layer_num,batch_size,epochs)


weights_hidden, weights_output=initialize_weights(1000)

learning_rate=0.01

epochs=50

batch_size=10

layer_num=1000


train2(Train_input,Train_label,Test_input ,Test_label,weights_output,
weights_hidden,activation_hidden_1, activation_output_1,
learning_rate,layer_num,batch_size,epochs)


weights_hidden, weights_output=initialize_weights(1000)

learning_rate=0.01

epochs=50

batch_size=50

layer_num=1000


train2(Train_input,Train_label,Test_input ,Test_label,weights_output,
weights_hidden,activation_hidden_1, activation_output_1,
learning_rate,layer_num,batch_size,epochs)


weights_hidden, weights_output=initialize_weights(1000)

learning_rate=0.01
```

```
epochs=50

batch_size=100

layer_num=1000


train2(Train_input,Train_label,Test_input ,Test_label,weights_output,
weights_hidden,activation_hidden_1, activation_output_1,
learning_rate,layer_num,batch_size,epochs)


def train3(inputs, labels,test_images ,test_label,weights_hidden_output, weights_input_hidden,
activation_hidden, activation_output, learning_rate, layer_num, batch_size, epochs,lambda_coef):

    trainingError = 0.0

    startTime = time.time()


    for epoch in range(epochs):

        cumulativeError = 0.0


        for i in range(batch_size):

            flattened_data = np.reshape(inputs[i], (784, 1)) / 255

            error, weights_hidden_output, weights_input_hidden =
backpropagation_sigmoid2(flattened_data, labels[i], weights_hidden_output, weights_input_hidden,
activation_hidden, activation_output, learning_rate, layer_num,lambda_coef)

            cumulativeError += np.sum(error**2 * 0.5)


        epochErrorMean = cumulativeError / batch_size

        trainingError += epochErrorMean


        print("Epoch Completed: ", epoch + 1, "\nMean Squares Error of Epoch: ", epochErrorMean)


    trainingErrorMean = trainingError / epochs

    stopTime = time.time()

    print("Training Completed!\nCPU Time: ", stopTime - startTime, " seconds", "\nOverall Error:",
trainingErrorMean)
```

```
    cumulativeError_2 = 0.0
    numOfMissClassification = 0

    for i in range(10000):
        testData = np.reshape(test_images[i], (784, 1)) / 255
        layer_output,final_output=forward_pass(testData, weights_input_hidden,
weights_hidden_output,activation_hidden, activation_output)
        desired=desired_output_tanh(test_label[i],0)
        error_2=find_error(final_output,desired)
        cumulativeError_2 += np.sum(error_2**2 * 0.5)

        predicted = np.argmax(final_output)

        if test_label[i] != predicted:
            numOfMissClassification += 1

    misclasificationPercentage = numOfMissClassification / 10000 * 100

    testSetErrorMean = cumulativeError_2 / 10000

    print("Number Of Misclasification:",numOfMissClassification,"\nError Percentage:
%",misclasificationPercentage,"\nTest Data Mean Square Error:",testSetErrorMean)

weights_hidden, weights_output=initialize_weights(300)
activation_hidden_1= relu_activation
activation_output_1= sigmoid_activation
batch_size=1250
learning_rate=0.01
lambda_coef=0.001
epochs=50
layer_num=300
```

```
train3(Train_input,Train_label,Test_input ,Test_label,weights_output,
weights_hidden,activation_hidden_1, activation_output_1,
learning_rate,layer_num,batch_size,epochs,lambda_coef)


weights_hidden, weights_output=initialize_weights(1000)

activation_hidden_1= relu_activation

activation_output_1= sigmoid_activation

batch_size=1250

learning_rate=0.01

lambda_coef=0.001

epochs=50

layer_num=1000


train3(Train_input,Train_label,Test_input ,Test_label,weights_output,
weights_hidden,activation_hidden_1, activation_output_1,
learning_rate,layer_num,batch_size,epochs,lambda_coef)


weights_hidden, weights_output=initialize_weights(300)

learning_rate=0.01

epochs=50

batch_size=60000

layer_num=300


train2(Train_input,Train_label,Test_input ,Test_label,weights_output,
weights_hidden,activation_hidden_1, activation_output_1,
learning_rate,layer_num,batch_size,epochs)


weights_hidden, weights_output=initialize_weights(1000)

activation_hidden_1= relu_activation

activation_output_1= sigmoid_activation

batch_size=100

learning_rate=0.01

lambda_coef=0.001
```

epochs=50

layer_num=1000

```
train3(Train_input,Train_label,Test_input ,Test_label,weights_output,
weights_hidden,activation_hidden_1, activation_output_1,
learning_rate,layer_num,batch_size,epochs,lambda_coef)
```