CS 454

Introduction to Machine Learning and Artificial Neural Networks

Prof. Dr. Ethem Alpaydın

Fall 2021

Homework #3

Neural Networks

by

Abdullah Saydemir

S014646

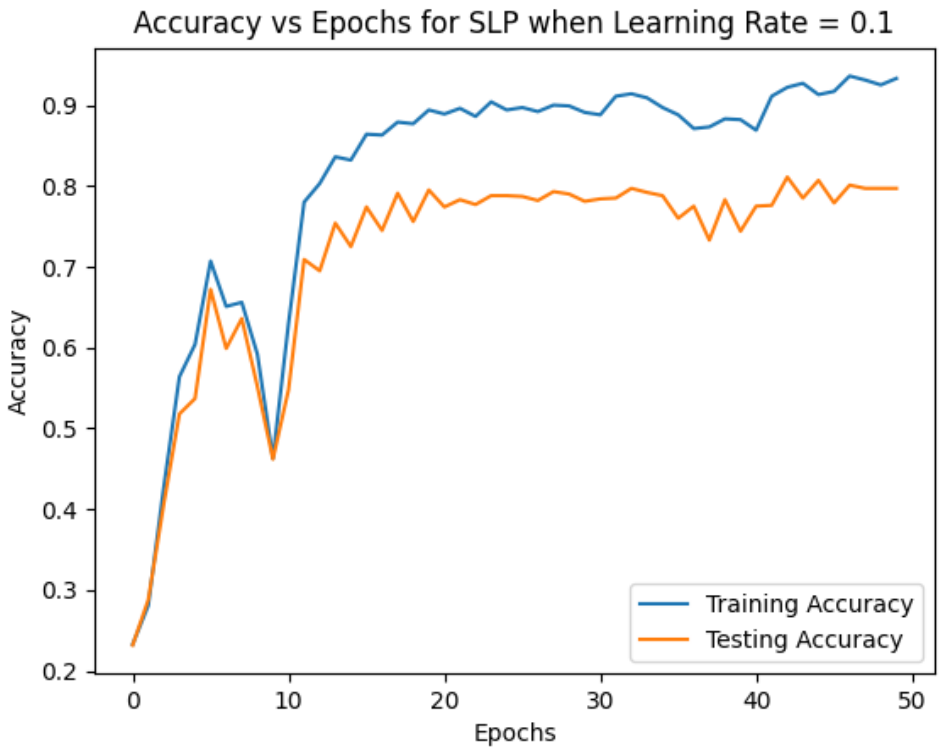
December 5, 2021

Part 1 - Single-Layer Perceptron

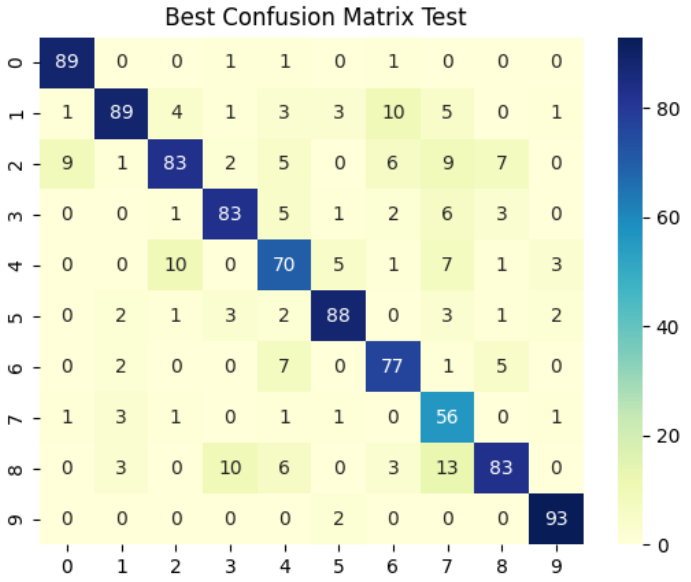
I tested the input with learning rates `0.01, 0.025, 0.05, 0.1` and `0.2`. I obtained the best result when my learning rate was `0.1`. Log files for the other rates can be checked from the GitHub repository. If you want to reproduce the experiment, you can run *process.sh* with the dependencies. Output will be saved to `./log/single_layer_log_${learning_rate}.txt`. One important note is that I did not shuffle the images in each epoch because of a bug I could not solve, therefore the accuracies should be a little bit lower.

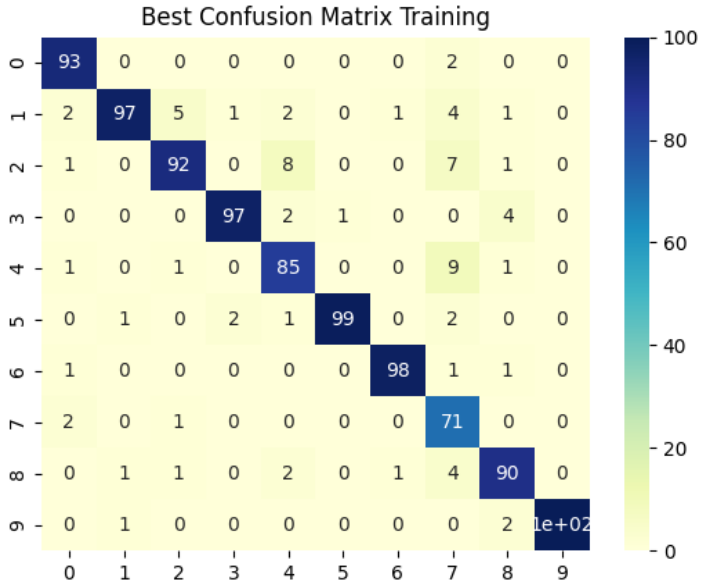
a. Implementation is provided below and will be available through this link after the deadline. print functions that output the y-matrices are commented out not to create a mess on the console.

b. Plots of the accuracies of the training and test dataset are as follows.

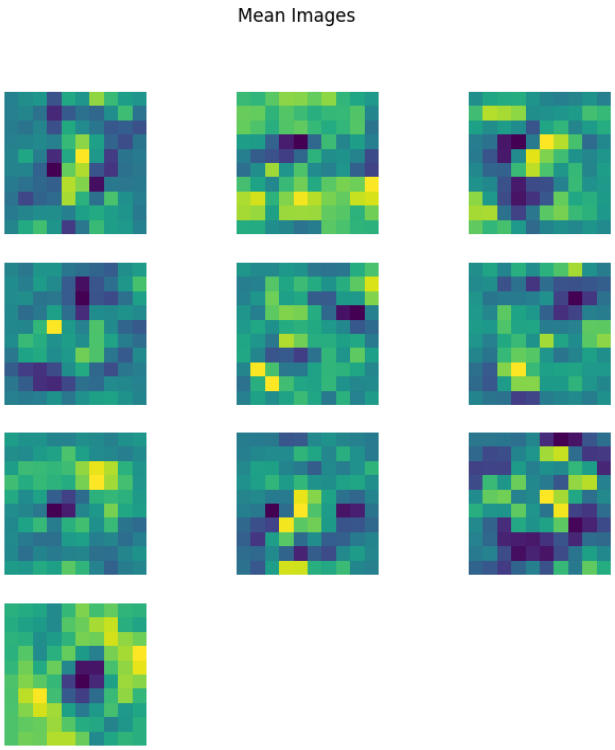


c. Confusion matrices for the training and test dataset are as follows. These are obtained from the epochs with the highest accuracy. Rows represent the predictions while columns represent the actual labels.





d. Weight vectors for each label are below. Thanks to Mahir for providing the printing code.

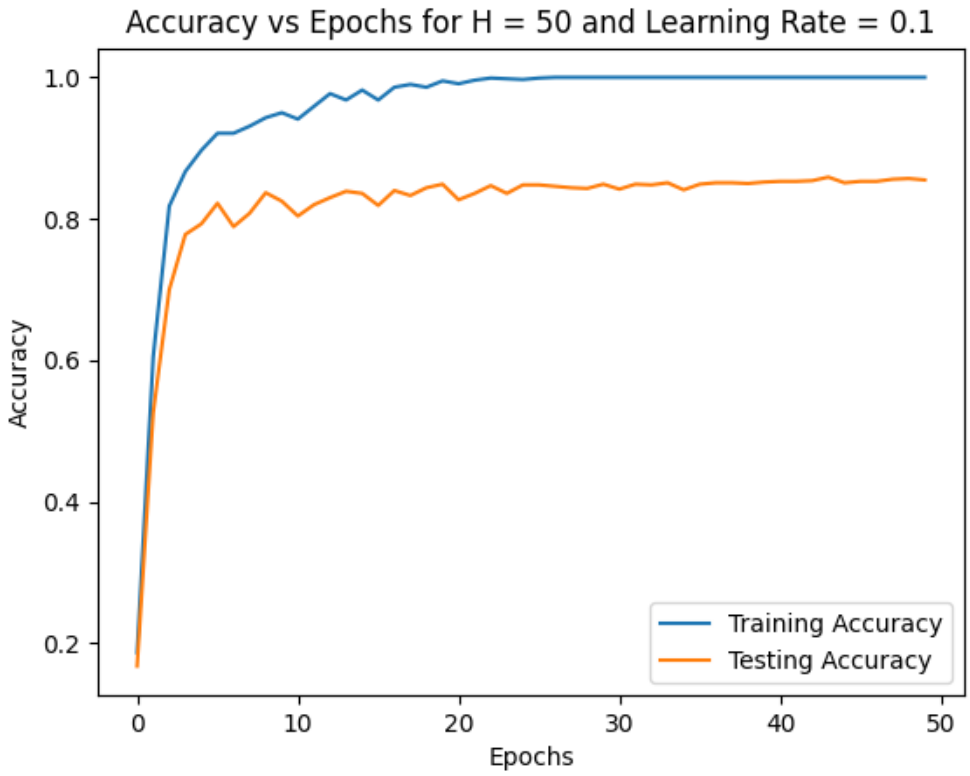


Part 2 - Multi-Layer Perceptron

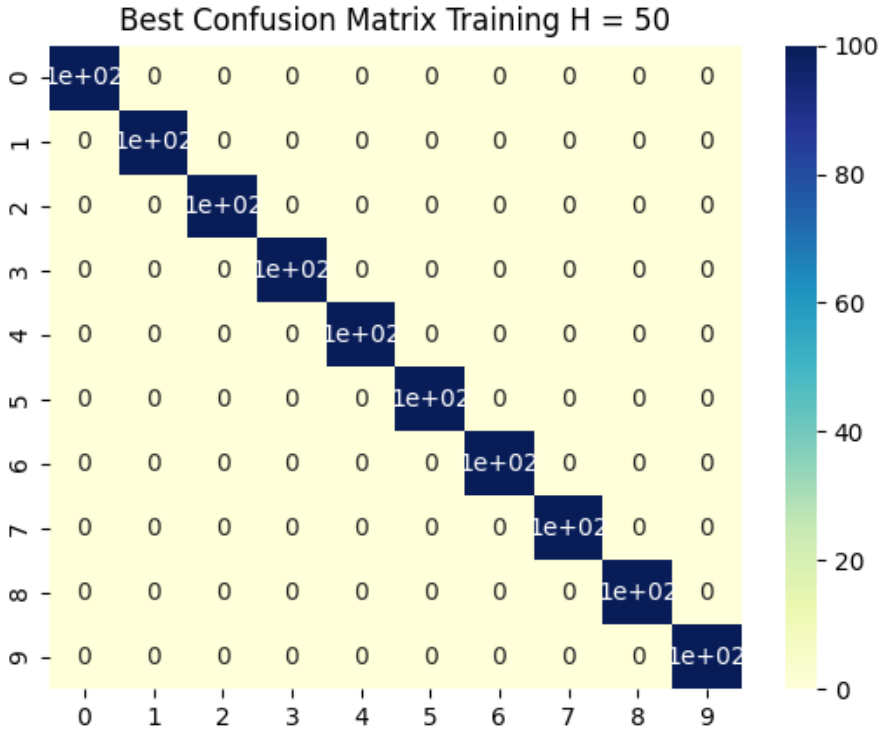
I tested the input with learning rates `0.01` `0.025` `0.05` `0.1` `0.2` `0.001` `0.0005` and `0.0001`. I obtained the best result when my learning rate was `0.1`. Log files for the other rates can also be checked from the GitHub repository. If you want to reproduce the experiment, you can run *process.sh* with the dependencies. Output will be saved to `./log/multi_layer_log_${learning_rate}.txt`

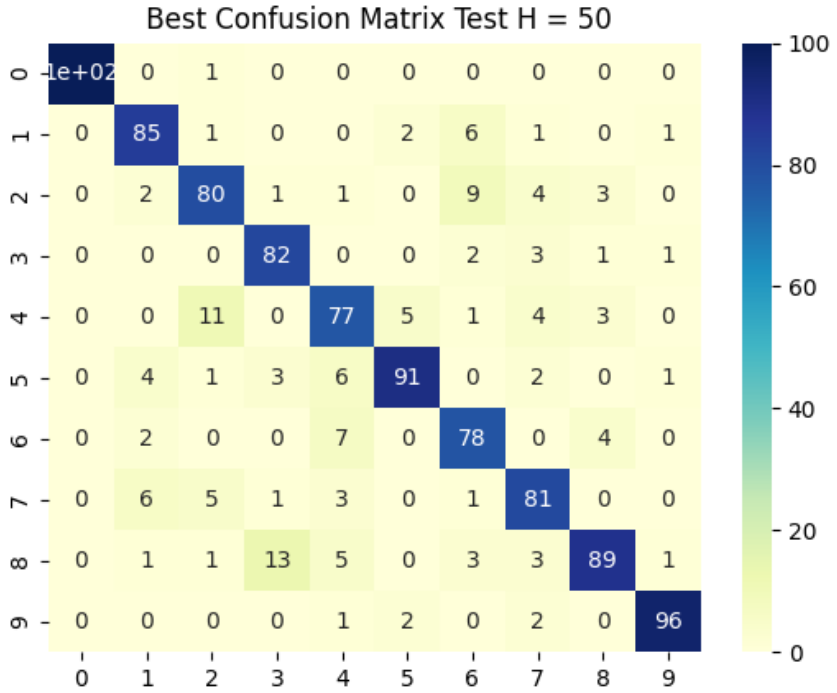
a. Implementation is provided below and will be available through this link after the deadline.

b. Plots of the accuracies of the training and test dataset are as follows. Plots are obtained from the highest accuracy epochs when H = 50. Plots for each number of hidden units are inside `./graph/` directory in the GitHub repo.



c. Confusion matrices for the training and test dataset are as follows. These are obtained from the epochs with the highest accuracy when H = 50. Plots for each number of hidden units are inside `./graph/` directory in the GitHub repo. Rows represent the predictions while columns represent the actual label.


Best Confusion Matrix Training H = 50
0 1e+02 0 0 0 0 0 0 0 0 0
1 0 1e+02 0 0 0 0 0 0 0 0
2 0 0 1e+02 0 0 0 0 0 0 0
3 0 0 0 1e+02 0 0 0 0 0 0
4 0 0 0 0 1e+02 0 0 0 0 0
5 0 0 0 0 0 1e+02 0 0 0 0
6 0 0 0 0 0 0 1e+02 0 0 0
7 0 0 0 0 0 0 0 1e+02 0 0
8 0 0 0 0 0 0 0 0 1e+02 0
9 0 0 0 0 0 0 0 0 0 1e+02
0 1 2 3 4 5 6 7 8 9
100
80
60
40
20
0


Best Confusion Matrix Test H = 50
0 1e+02 0 1 0 0 0 0 0 0 0
1 0 85 1 0 0 2 6 1 0 1
2 0 2 80 1 1 0 9 4 3 0
3 0 0 0 82 0 0 2 3 1 1
4 0 0 11 0 77 5 1 4 3 0
5 0 4 1 3 6 91 0 2 0 1
6 0 2 0 0 7 0 78 0 4 0
7 0 6 5 1 3 0 1 81 0 0
8 0 1 1 13 5 0 3 3 89 1
9 0 0 0 0 1 2 0 2 0 96
0 1 2 3 4 5 6 7 8 9
100
80
60
40
20
0

Observations

Shuffling enhances the learning of the perceptrons. I couldn't use it in the single-layer perceptron but for MLP it gave critically better results. For example when H = 5, MLP without shuffling gave 0.3 accuracy at max. On the other hand, MLP with shuffling provided little more than 0.5, which is close to double compared to MLP without shuffling. For larger H, the difference is still preserved but it becomes less important. Therefore, I know that I would get better results for SLP if I could utilize shuffling.

Weights in the single-layer perceptron show how the perceptron "percepts" the digits. It tries to find similar digits by looking at the new image and its perceptions of the digits. If the images used in training were digitally printed, then noise would be close to zero and we could see the numbers clearly from the weights.

Multi-Layer perceptron performs better than single layer for H values greater than 10. MLP memorizes the training dataset after a certain number of epochs and the accuracy flattens out at 1.0. For smaller H values, MLP cannot fully perform and give worse results than SLP.

Training accuracies are most of the time better than the test accuracies for both algorithms.

What I learned

- Learning rate is somewhat important. If you give a large learning rate, then the accuracy fluctuates after some point. Similarly, if the learning rate is low, then the training ends before the weights are optimized.
- You have to be sharp. I missed shifting the indices to the right in multi-layer perceptron code and performance was even worse than single layer perceptron.
- I didn't know we should forward the images once more after we update the tables. Pseudocode had only the first part (forward & backward), and I was predicting before updating the tables. I fixed that after attending the office hours.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sys
import seaborn as sn

# Taken from Mahir, the TA
def plot_mean_images(weights):
    # Creating 4*3 subplots
    fig, axes = plt.subplots(4, 3)
    # Set the height of plat 8px*8px
    fig.set_figheight(8)
    fig.set_figwidth(8)
    fig.suptitle('Mean Images')
    # For each subplot run the code inside loop
    for label in range(12):
        # If the subplot index is a label (0,1,2...9)
        if label<10:
            axes[label//3][label%3].imshow(weights[:,label].reshape(10,10),)
        # Do not show the axes of subplots
        axes[label//3][label%3].axis('off')
    # Showing the plot
    plt.show()


def softmax(x):
    e_x = np.exp(x - np.max(x))
    return e_x / e_x.sum()

def one_hot_encode(y):
    return np.transpose(np.eye(10)[y-1])

def accuracy_confusion_matrix(x):
    correct = 0
    total   = 0
    for i in range(len(x)):
        for j in range(len(x[i])):
            total += x[i][j]
            if i == j:
                correct += x[i][j]
    return correct/total

learning_rate = 0.1
K = 10
d = 100
E = 50

if len(sys.argv) == 2:
    learning_rate = float(sys.argv[1])

matrix = pd.read_csv('training.csv', header=None, skiprows=1)

R = matrix.iloc[:,0]
X = matrix.iloc[:,1:] / 255
N = matrix.shape[0]

confusion_matrix_training = np.zeros((K,K))
confusion_matrix_test = np.zeros((K,K))
w_matrix = np.random.uniform(low=-0.01, high=0.01, size=(d+1,K))
y_matrix = np.zeros((N,K))

matrix_test = pd.read_csv('testing.csv', header=None, skiprows=1)
R_test = matrix_test.iloc[:,0]
X_test = matrix_test.iloc[:,1:] / 255
N_test = matrix_test.shape[0]
y_matrix_test = np.zeros((N_test,K))

accuracy_best = 0
w_matrix_best = np.zeros((d+1,K))
confusion_matrix_training_best = np.zeros((K,K))
confusion_matrix_test_best = np.zeros((K,K))
```

```
print("-----------------------------------------------------------------------")
print("K = ", K)
print("E = ", E)
print("d = ", d)
print("N = ", N)
print("N_test = ", N_test)
print("Learning rate = ", learning_rate)
print("Starting the process...\n")


training_accuracies = []
test_accuracies = []
for epochs in range(E):

    delta_w = np.zeros(w_matrix.shape)

    for t in range(N):
        x_t = X.iloc[t,:]
        x_t = np.append(x_t, 1)
        x_t = np.transpose(x_t)

        o = np.zeros(K)
        for i in range(K):
            o[i] = np.dot(w_matrix[:,i], x_t)

        y_matrix[t] = softmax(o)
        r_t = one_hot_encode(R.iloc[t])

        for i in range(K):
            for j in range(d+1):
                delta_w[j,i] += (r_t[i] - y_matrix[t][i])  * x_t[j]

        for i in range(K):
            for j in range(d+1):
                w_matrix[j,i] += delta_w[j,i] * learning_rate

    for t in range(N):
        x_t = X.iloc[t,:]
        x_t = np.append(x_t, 1)
        x_t = np.transpose(x_t)

        o = np.zeros(K, dtype=np.float64)
        for i in range(K):
            o[i] = np.dot(w_matrix[:,i], x_t)

        y_matrix[t] = softmax(o)

    for t in range(N):
        x_t_test = X_test.iloc[t,:]
        x_t_test = np.append(x_t_test, 1)
        x_t_test = np.transpose(x_t_test)

        o = np.zeros(K, dtype=np.float64)
        for i in range(K):
            o[i] = np.dot(w_matrix[:,i], x_t_test)

        y_matrix_test[t] = softmax(o)


    confusion_matrix_training = np.zeros((K,K))
    confusion_matrix_test = np.zeros((K,K))

    for t in range(N):
        confusion_matrix_training[np.argmax(y_matrix[t]),R.iloc[t]-1]       += 1
    for t in range(N_test):
        confusion_matrix_test[np.argmax(y_matrix_test[t]),R_test.iloc[t]-1] += 1

    accuracy_training = accuracy_confusion_matrix(confusion_matrix_training)
    accuracy_test     = accuracy_confusion_matrix(confusion_matrix_test)
    test_accuracies.append(accuracy_test)
    training_accuracies.append(accuracy_training)

    print("Epoch: ", epochs+1, "Training Accuracy: ", accuracy_training, "Testing Ac
```

```
curacy: ", accuracy_test)

    if accuracy_test > accuracy_best:
        accuracy_best = accuracy_test
        w_matrix_best = w_matrix
        confusion_matrix_test_best     = confusion_matrix_test
        confusion_matrix_training_best = confusion_matrix_training


# print("\nConfusion Matrix Training:")
# print(*confusion_matrix_training_best, sep='\n', end="\n\n")


# print("Confusion Matrix Test:")
# print(*confusion_matrix_test_best, sep='\n', end="\n\n")
# print("Best Accuracy: ", accuracy_best)

df_cm = pd.DataFrame(confusion_matrix_training_best, index = [i for i in range(10)],
                columns = [i for i in range(10)])

plt.title('Best Confusion Matrix Training')
sn.heatmap(df_cm, annot=True, cmap="YlGnBu")
plt.savefig('./graphs/confusion_matrix_training_single.png')
#plt.show()
plt.clf()

#print("Confusion Matrix Test H = " + str(H) + " :")
#print(*confusion_matrix_test_best, sep='\n', end="\n\n")

df_cm = pd.DataFrame(confusion_matrix_test_best, index = [i for i in range(10)],
                columns = [i for i in range(10)])

plt.title(' Best Confusion Matrix Test')
sn.heatmap(df_cm, annot=True, cmap="YlGnBu")
plt.savefig('./graphs/confusion_matrix_test_single_.png')
#plt.show()
plt.clf()

#plot_mean_images(w_matrix_best[:-1,:])
#print("10 Dimensional Probabilities :", y_matrix)


plt.title("Accuracy vs Epochs for SLP when Learning Rate = " + str(learning_rate))
plt.plot(training_accuracies, label='Training Accuracy')
plt.plot(test_accuracies, label='Testing Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.savefig("./graphs/Accuracy_vs_Epochs_Single_LR_" + str(learning_rate) + ".png")
#plt.show()
```

```
from os import sep
import numpy as np
import pandas as pd
import random
import matplotlib.pyplot as plt
import sys
import seaborn as sn

# Taken from Mahir, the TA
def plot_mean_images(weights):
    # Creating 4*3 subplots
    fig, axes = plt.subplots(4, 3)
    # Set the height of plat 8px*8px
    fig.set_figheight(8)
    fig.set_figwidth(8)
    fig.suptitle('Mean Images')
    # For each subplot run the code inside loop
    for label in range(12):
        # If the subplot index is a label (0,1,2...9)
        if label<10:
            axes[label//3][label%3].imshow(weights[:,label].reshape(10,10),)
        # Do not show the axes of subplots
        axes[label//3][label%3].axis('off')
    # Showing the plot
    plt.show()


def accuracy_confusion_matrix(x):
    correct = 0
    total   = 0
    for i in range(len(x)):
        for j in range(len(x[i])):
            total += x[i][j]
            if i == j:
                correct += x[i][j]
    return correct/total


def one_hot_encode(y):
    return np.transpose(np.eye(10)[y-1])


def one_hot_decode(y):
    return np.argmax(y) + 1


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


def softmax(x):
    e_x = np.exp(x - np.max(x))
    return e_x / e_x.sum()


learning_rate = 0.1
K = 10
E = 50
d = 100

if len(sys.argv) == 2:
    learning_rate = float(sys.argv[1])

matrix = pd.read_csv('training.csv', header=None, skiprows=1)
R = matrix.iloc[:,0]
X = matrix.iloc[:,1:] / 255
N = X.shape[0]
y_matrix = np.zeros((N,K))

matrix_test = pd.read_csv('testing.csv', header=None, skiprows=1)
R_test = matrix_test.iloc[:,0]
X_test = matrix_test.iloc[:,1:] / 255
N_test = matrix_test.shape[0]
```

```
y_matrix_test = np.zeros((N_test,K))

Hs              = [5, 10, 25, 50, 75]
best_H_accuracy = 0
best_H          = -1
best_H_accuracies = []

print("K = ", K)
print("E = ", E)
print("d = ", d)
print("N = ", N)
print("N_test = ", N_test)
print("Learning rate = ", learning_rate)
print("H values are: ", *Hs)

print("Starting the process...\n")
print("-----------------------------------------------------------------")

for H in Hs:
    accuracy_best = 0
    w_matrix_best = np.zeros((d+1,H))
    confusion_matrix_training_best = np.zeros((K,K))
    confusion_matrix_test_best     = np.zeros((K,K))

    training_accuracies = []
    test_accuracies     = []

    v_matrix = np.random.uniform(low=-0.01, high=0.01, size=(H+1, K))
    w_matrix = np.random.uniform(low=-0.01, high=0.01, size=(d+1 ,H))
    for epochs in range(E):
        random_list = list(range(N))
        random.shuffle(random_list)

        for instance in random_list:
            x_t = X.iloc[instance,:]
            x_t = np.append(1, x_t)
            x_t = np.transpose(x_t)

            z = np.zeros(H)

            for h in range(H):
                w_h_T  = np.transpose(w_matrix[:,h])
                z[h]   = np.dot(w_h_T, x_t)

            z = sigmoid(z)

            z = np.append(1, z)
            z = np.transpose(z)

            os = np.zeros(K)
            for i in range(K):
                os[i] = np.dot(np.transpose(v_matrix[:,i]), z)


            y_matrix[instance] = softmax(os)

            delta_v_matrix = np.zeros(v_matrix.shape)
            delta_w_matrix = np.zeros(w_matrix.shape)

            r_t = one_hot_encode(R.iloc[instance])

            for i in range(K):
                delta_v_matrix[:,i] = learning_rate*(r_t[i] - y_matrix[instance][i])
 * z

            for h in range(H):
                summa = 0.0
                for i in range(K):
                    summa += (r_t[i] - y_matrix[instance][i])*v_matrix[h,i]
                delta_w_matrix[:,h] = learning_rate*summa*z[h+1]*(1-z[h+1])*x_t

            for i in range(K):
                v_matrix[:,i] += delta_v_matrix[:,i]
```

```
            for h in range(H):
                w_matrix[:,h] += delta_w_matrix[:,h]

        for instance in random_list:
            x_t = X.iloc[instance,:]
            x_t = np.append(1, x_t)
            x_t = np.transpose(x_t)

            z = np.zeros(H)
            for h in range(H):
                w_h_T  = np.transpose(w_matrix[:,h])
                z[h]   = np.dot(w_h_T, x_t)

            z = sigmoid(z)

            z = np.append(1, z)
            z = np.transpose(z)

            os = np.zeros(K)
            for i in range(K):
                os[i] = np.dot(np.transpose(v_matrix[:,i]), z)

            y_matrix[instance] = softmax(os)

        for instance in random_list:
            x_t_test = X_test.iloc[instance,:]
            x_t_test = np.append(1, x_t_test)
            x_t_test = np.transpose(x_t_test)

            z_test = np.zeros(H)
            for h in range(H):
                w_h_T  = np.transpose(w_matrix[:,h])
                z_test[h]   = np.dot(w_h_T, x_t_test)

            z_test = sigmoid(z_test)

            z_test = np.append(1, z_test)
            z_test = np.transpose(z_test)

            os = np.zeros(K)
            for i in range(K):
                os[i] = np.dot(np.transpose(v_matrix[:,i]), z_test)

            y_matrix_test[instance] = softmax(os)

        confusion_matrix_training = np.zeros((K,K))
        confusion_matrix_test     = np.zeros((K,K))

        for t in range(N):
            confusion_matrix_training[np.argmax(y_matrix[t])][R.iloc[t]-1] += 1
        for t in range(N_test):
            confusion_matrix_test[np.argmax(y_matrix_test[t])][R_test.iloc[t]-1] +=
1

        accuracy_training = accuracy_confusion_matrix(confusion_matrix_training)
        accuracy_test     = accuracy_confusion_matrix(confusion_matrix_test)
        test_accuracies.append(accuracy_test)
        training_accuracies.append(accuracy_training)

        print("H: ", H," Epoch: ", epochs+1, "Training Accuracy: ", accuracy_trainin
g, "Testing Accuracy: ", accuracy_test)

        if accuracy_test > accuracy_best:
            accuracy_best = accuracy_test
            w_matrix_best = w_matrix
            confusion_matrix_test_best     = confusion_matrix_test
            confusion_matrix_training_best = confusion_matrix_training
            best_H = H

    best_H_accuracies.append(accuracy_best)
    print("H: ", H, " Best Accuracy: ", accuracy_best)
    print("-----------------------------------------------------------------")
```

```
    #print("10 Dimensional Probabilities (Training) :", y_matrix)
    #print("10 Dimensional Probabilities (Testing) :", y_matrix_test)

    #print("\nConfusion Matrix Training H = " + str(H) + " :")
    #print(*confusion_matrix_training_best, sep='\n', end="\n\n")

    df_cm = pd.DataFrame(confusion_matrix_training_best, index = [i for i in range(1
0)],
                  columns = [i for i in range(10)])
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title('Best Confusion Matrix Training H = ' + str(H))
    sn.heatmap(df_cm, annot=True, cmap="YlGnBu")
    plt.savefig('./graphs/confusion_matrix_training_H_' + str(H) + '.png')
    #plt.show()
    plt.clf()



    #print("Confusion Matrix Test H = " + str(H) + " :")
    #print(*confusion_matrix_test_best, sep='\n', end="\n\n")

    df_cm = pd.DataFrame(confusion_matrix_test_best, index = [i for i in range(10)],
                  columns = [i for i in range(10)])
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title(' Best Confusion Matrix Test H = ' + str(H))
    sn.heatmap(df_cm, annot=True, cmap="YlGnBu")
    plt.savefig('./graphs/confusion_matrix_test_H_' + str(H) + '.png')
    #plt.show()
    plt.clf()


    plt.title("Accuracy vs Epochs for H = " + str(H) + " and Learning Rate = " + str
(learning_rate))
    plt.plot(training_accuracies, label='Training Accuracy')
    plt.plot(test_accuracies, label='Testing Accuracy')
    plt.xlabel("Epochs")
    plt.ylabel("Accuracy")
    plt.legend(loc='lower right')
    #plt.show()
    plt.savefig("./graphs/Accuracy_vs_Epochs_H_" + str(H) + "_LR_" + str(learning_ra
te) + ".png")
    plt.clf()

print(*best_H_accuracies, sep = "\n")
```