CS 559 Homework 3

Huiyang Zhao
655490960

Codes and figures generated are attached after comments.

Output of the program

```
------Training------
n : 50, lr: 1, ε : 0, epochs: 15, errors: 0
------Testing------
errors : 4822, error rate: 0.4822
------Training------
n : 1000, lr: 1, ε : 0, epochs: 332, errors: 0
------Testing------
errors : 1687, error rate: 0.1687
------Training------
n : 60000, lr: 1, ε : 0, epochs: 500, errors: 11644
------Testing------
errors : 1438, error rate: 0.1438
------Training------
n : 60000, lr: 0.1, ε : 0.125, epochs: 248, errors: 7329
------Testing------
errors : 1327, error rate: 0.1327
------Training------
n : 60000, lr: 0.1, ε : 0.125, epochs: 83, errors: 7341
------Testing------
errors : 1388, error rate: 0.1388
------Training------
n : 60000, lr: 0.1, ε : 0.125, epochs: 331, errors: 7385
------Testing------
errors : 1446, error rate: 0.1446
```

Results and Comments

(f) n = 50, η = 1, ε = 0

The picture is attached as Figure 1.
Epochs taken: 15
Percentage of misclassified test samples: 0.4822.
Discrepancy: the training converges on the training and has 0 error, while there are 48.22% errors when testing the network on the test set.
Explain: the discrepancy occurred because the training set is too small. The network only learned on a small set of data which cannot cover the entire dataset's attributes. As a result, in the process the gained weight works well on the given training set but cannot work well in general.

(g) n = 1000, η = 1, ε = 0

The picture is attached as Figure 2.
Epochs taken: 332
Percentage of misclassified test samples: 0.1687.
Compare: the training converges on the training and has 0 error, while there are 16.87% errors when testing the network on the test set. The performance on the test set is much better than when n = 50.
Explain: with the enlargement of training set, the network now can learn through a large set that can cover a considerable number of attributes for the dataset, which results in a more accurate weights assignment for the neural network. We can see that a larger training set makes the neural network better.

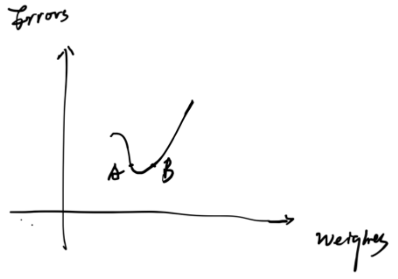
(h) n = 60000, η = 1, ε = 0

The picture is attached as Figure 3.
Epochs taken: After 500 epochs, the training still cannot converge to 0.
Percentage of misclassified test samples: 0. 1438.
Comment:
The number of errors get reduced quickly from close to 10000 to between 10000 and 20000.
Although it won't converge on the training set, we have a better performance on the test set. That means our trained network has a considerable performance.
However, the number of errors fluctuates a lot between 10000 to 20000, which means we cannot find a relatively optimal assignment of weights.

(i) My chosen η = 0.1, ε = 0.125

The picture is attached as Figure 3, Figure 4 and Figure 5.
Observations:
The training converges not to 0 but at some points between 10000 to 20000, since I want to make it better, I choose ε = 7500 / 60000 = 0.125.
On the other hand, why is the number of misclassifications reluctant to continue decreasing? My guess is that at some point, the learning rate η is too big that the weight fluctuates between two points A and B around the better weight assignment and cannot move to that, as shown below:
Below are results to 3 times training on different initial weights.

```
------Training------
n : 60000, lr: 0.1, ε : 0.125, epochs: 248, errors: 7329
------Testing------
errors : 1327, error rate: 0.1327
------Training------
n : 60000, lr: 0.1, ε : 0.125, epochs: 83, errors: 7341
------Testing------
errors : 1388, error rate: 0.1388
------Training------
n : 60000, lr: 0.1, ε : 0.125, epochs: 331, errors: 7385
------Testing------
errors : 1446, error rate: 0.1446
```



Comment:
Training is a process that is influenced a lot by random facts. We can see that with different initial weights, the training requires different number of epochs to converge, and the final performance is also different. On the other hand, my guess might be correct, since the error rate is reduced after η is set to 0.1, which means the network has better performance, and the fluctuation scope become smaller. If that's true, changing learning rate should be a good strategy when training a network.

```python
# CS559 Neural Network
# Huiyang Zhao
# UIN 655490960


import numpy as np
import matplotlib.pyplot as plt

seed = 655490960
np.random.seed(seed)


# Endorsed by instructor. Used for reading .gz files.
def read_idx(filename):
    import gzip
    import struct
    with gzip.open(filename, 'rb') as f:
        zero, data_type, dims = struct.unpack('>HBB', f.read(4))
        shape = tuple(struct.unpack('>I', f.read(4))[0] for d in range(dims))
        return np.frombuffer(f.read(), dtype=np.uint8).reshape(shape)


# step activation function u(·)
def step(x):
    return np.heaviside(x, 1)  # if x<0 output 0


if __name__ == '__main__':
    train_inputs = read_idx("train-images-idx3-ubyte.gz")
    train_labels = read_idx("train-labels-idx1-ubyte.gz")
    test_input = read_idx("t10k-images-idx3-ubyte.gz")
    test_labels = read_idx("t10k-labels-idx1-ubyte.gz")

    # (d) Build and train the network.
    # 0)Given η, ε, n:
    lr = 1
    lr_set = [1, 0.1, 0.1, 0.1]
    lr_flag = 0
    eps = 0
    # n = 50
    N = [50, 1000, 60000, 60000, 60000, 60000]
    # 1) Initialize W ∈ R10×784 randomly.
    W_init = np.random.uniform(-1, 1, (10, 784))

    for n in N:
        if n == 60000:
            lr = lr_set[lr_flag]
            if lr_flag > 0:
                eps = 0.125
                np.random.seed(seed + lr_flag)
                W_init = np.random.uniform(-1, 1, (10, 784))
            lr_flag += 1
        # 2) Initialize epoch = 0.
        epoch = 0
        # 3) Initialize errors(epoch) = 0, for epoch = 0, 1, ....
        array_errors = []
        W = W_init
        flag = True
        print('------Training------')
        while flag:
            num_errors = 0
            array_x = []
            array_v = []
            # 3.1.1 Count the misclassification errors.
            for i in range(n):
                x_i = train_inputs[i].reshape((784, 1))
                array_x.append(x_i)
                # 3.1.1.1 Calculate the induced local fields with the current training
sample and weights.
                v = np.matmul(W, x_i)
                array_v.append(v)
                # print(v)
                # 3.1.1.2 Choose the output neuron with the largest induced local
field.
                j = np.argmax(v)
                # 3.1.1.3 If j is not the same as the input label, then errors(epoch)
← errors(epoch) + 1.
                if j != train_labels[i]:
                    num_errors += 1

            # 3.1.3 update the weights.
            for i in range(n):
                d_xi = np.zeros((10, 1))
                d_xi[train_labels[i]] = 1
                W = W + lr * np.matmul((d_xi - step(array_v[i])),
array_x[i].transpose())

            # print('Epoch: {}, errors: {}, ε: {}'.format(epoch, num_errors, eps))
            # 3.1.2 epoch ← epoch + 1.
            epoch += 1
            array_errors.append(num_errors)

            if num_errors / n <= eps or epoch >= 500:
                flag = False

        print('n : {}, lr: {}, ε : {}, epochs: {}, errors: {}'.format(n, lr, eps,
epoch, num_errors))
        # (f) Plot the epoch number vs.the number of misclassification errors
        array_epoch = list(range(epoch))
        plt.figure()
        plt.title('Epochs vs. Misclassifications n = {}, η = {}, ε = {}'.format(n, lr,
eps))
        plt.xlabel('Epochs')
        plt.ylabel('Misclassifications')
        plt.plot(array_epoch, array_errors)
        plt.show()

        # (e) Test the network with test set.
        num_errors_test = 0
        for i in range(10000):
            x_i = test_input[i].reshape((784, 1))
            # 2.1 Calculate the induced local fields with the current training sample
and weights.
            v = np.matmul(W, x_i)
            # print(v)
            # 2.2 Choose the output neuron with the largest induced local field.
            j = np.argmax(v)
            # 2.3 If j is not the same as the input label, then errors ← errors + 1.
            if j != test_labels[i]:
                num_errors_test += 1
        # (f) Record the percentage of misclassified test samples
        print('------Testing------')
        print('errors : {}, error rate: {}'.format(num_errors_test, num_errors_test /
10000))
```

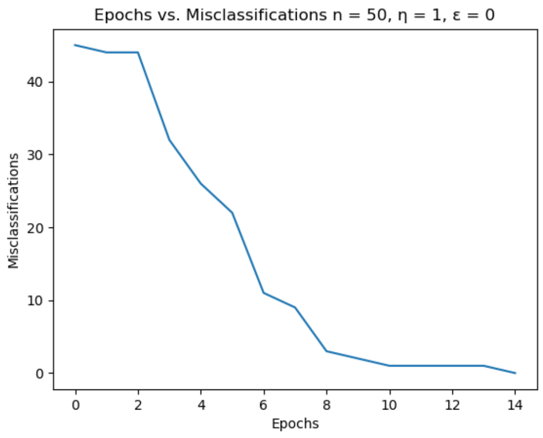


Figure 1

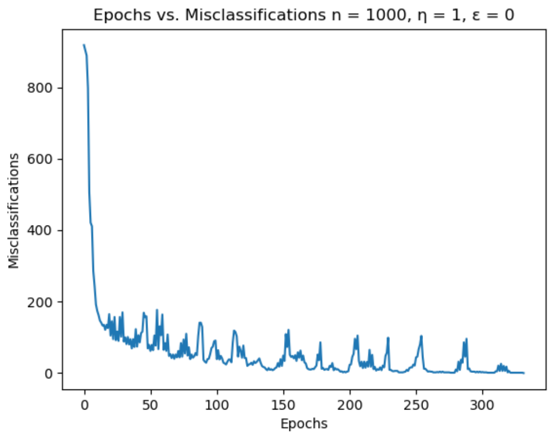


Figure 2

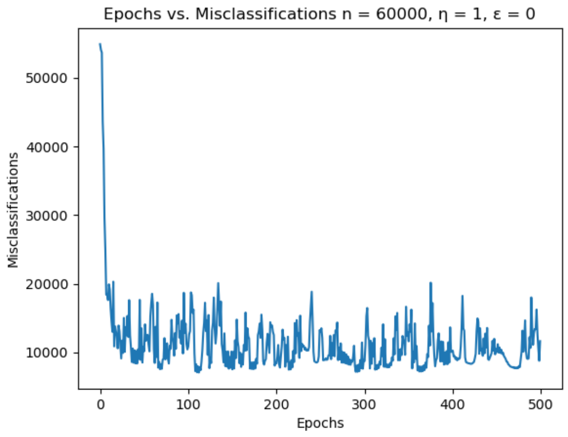


Figure 3

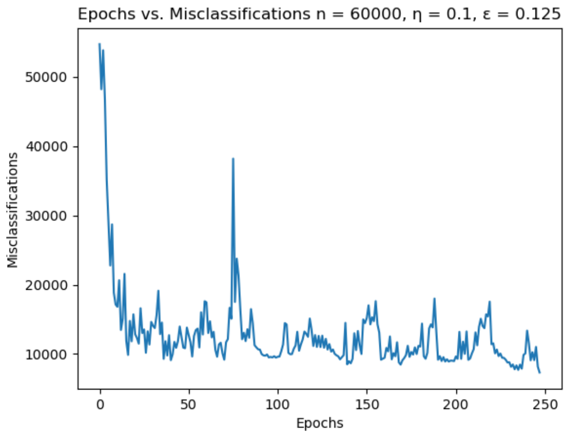


Figure 4

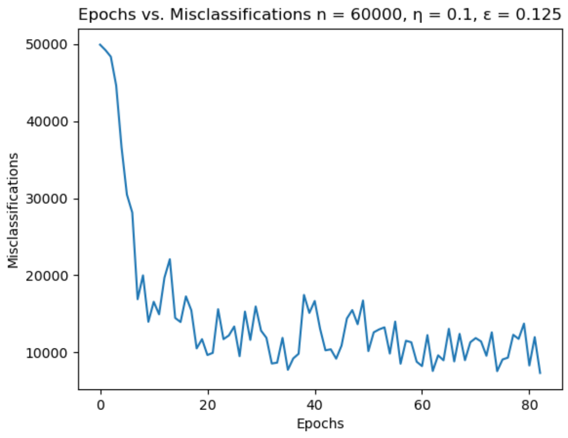


Figure 5

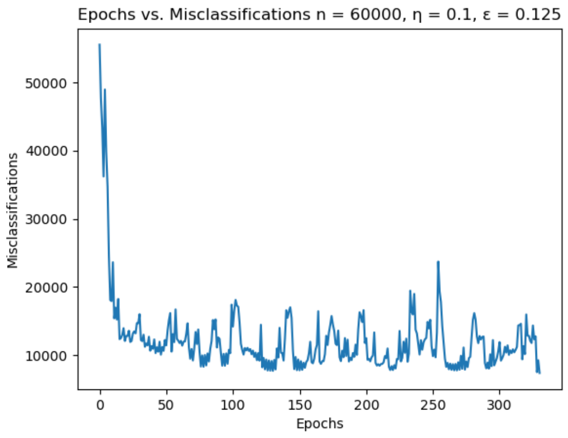


Figure 6