Sudarshan Pillai

11/30/20

Ali Minai-EECE 6036

HW # 5

1.1 System Description

Learning rate – 0.01

Learning rate time constant - 1000

Rule for choosing initial weights – random values between range of 0 to 1

Number of hidden layers – 1

Number of hidden neurons – 12x12

Variance (σ) – 6

Variance time constant – 1000 / $\log_{10}(\sigma)$ = 1285.0972089384688

Epochs – 20000

1.2 Results

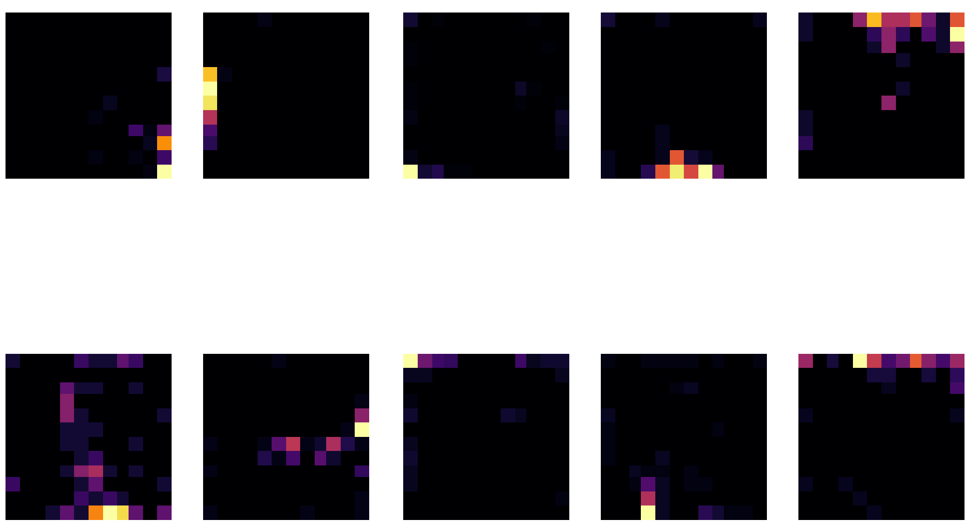

Figure 1.1 – Winning neuron per digit for the SOM

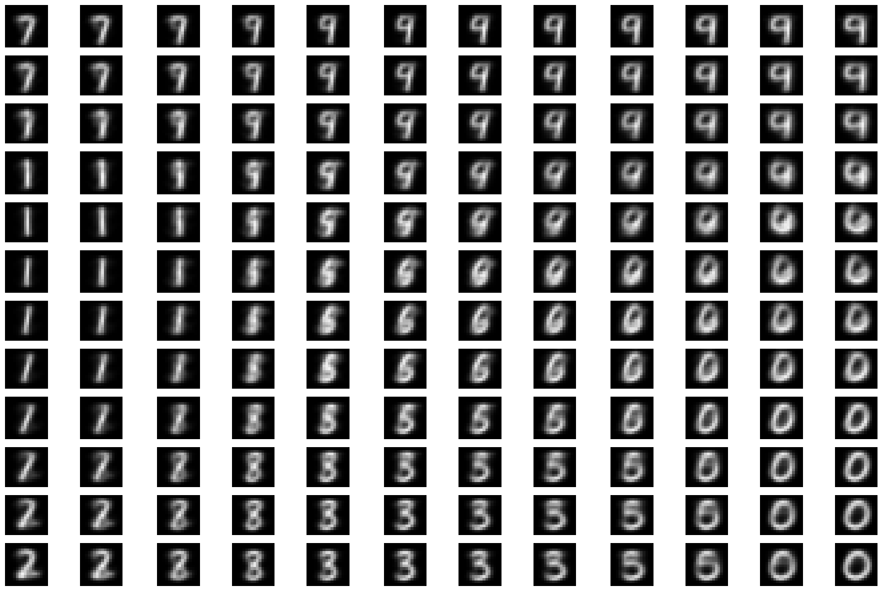

Figure 1.2 – Feature map of SOM learned by all the neurons

1.3 Analysis of Results

From figure 1.1, we can see that the weight map is used for mapping and grouping all the digits in the weight space. We can also see that each digit is identified by a specific region in the weight map. Finally, it is also bit interesting to see that the digits 4 and 9 have the same set of neurons firing in the weight space.

Figure 1.2 shows the features which are learned by all of the neurons in the 12x12 SOM feature map grid. The purpose of this gird is for each of these neurons to identify a particular digit. For example, in the middle to bottom right corner you can see that the neurons are being trained to identify the digit 8.

2.1 System Description

Learning rate – 0.01

Rule for choosing initial weights – random values between range of 0 to 1

Number of hidden layers – 1

Number of hidden neurons – 12x12

Variance (σ) – 6

Variance time constant – 1000 / $\log_{10}(\sigma)$ = 1285.0972089384688

Criterion to stop training – when error < 1% or 0.01

Output thresholds - 1 is >= 0.75, 0 is < 0.25

Epochs – 200

2.2 Results

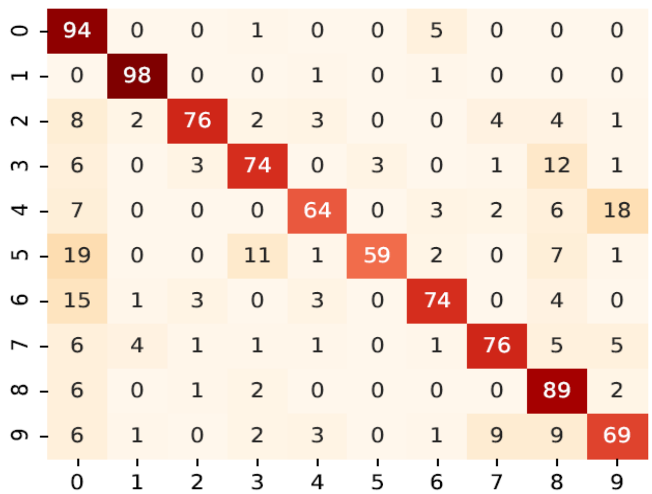

Figure 2.1 – Confusion matrix for SOFM weights

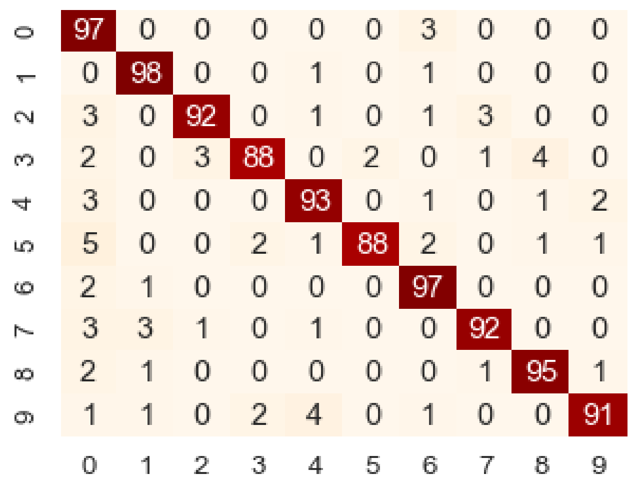

Figure 2.2 – Confusion matrix for classifier test set

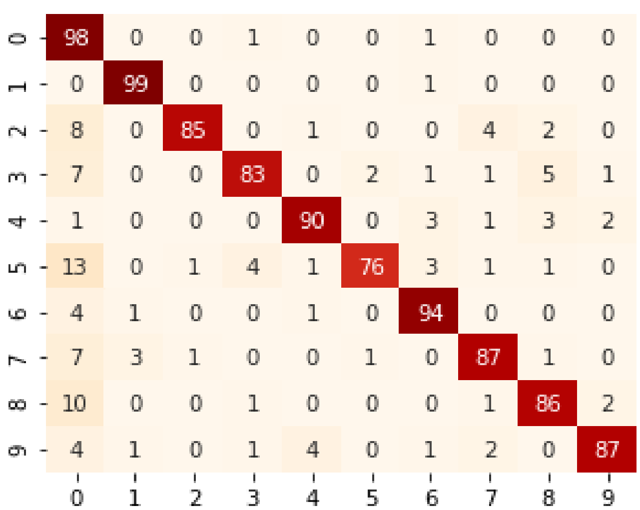

Figure 2.3 – Confusion matrix for autoencoder weights

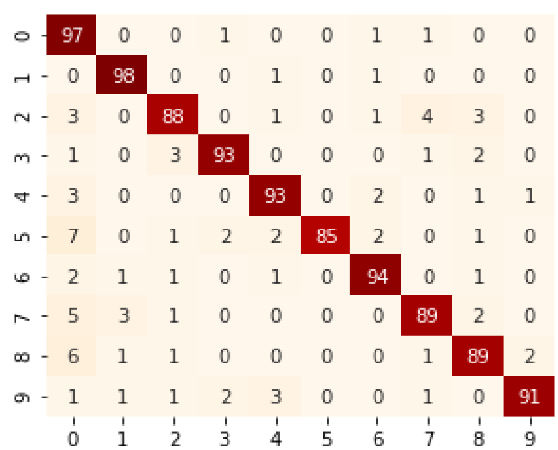

Figure 2.3 – Confusion matrix for autoencoder w/noise weights

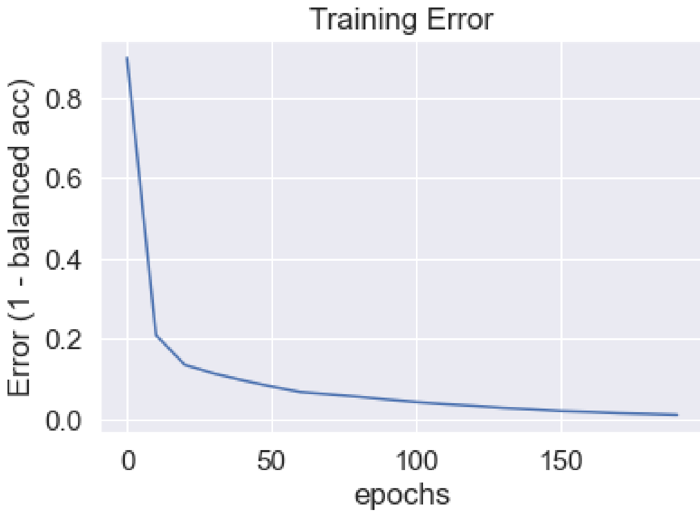

Figure 2.5 – training error for classifier

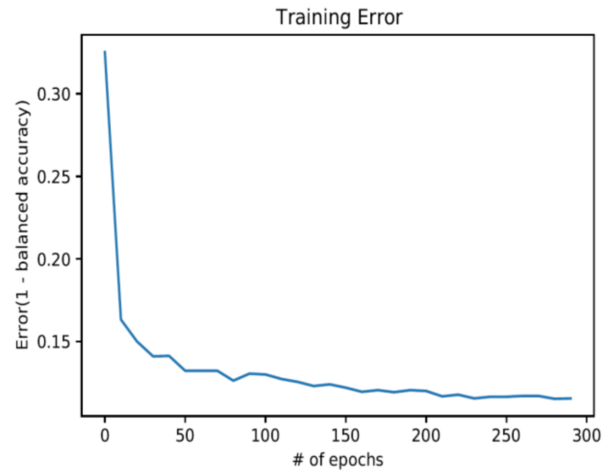

Figure 2.6 – training error for autoencoder

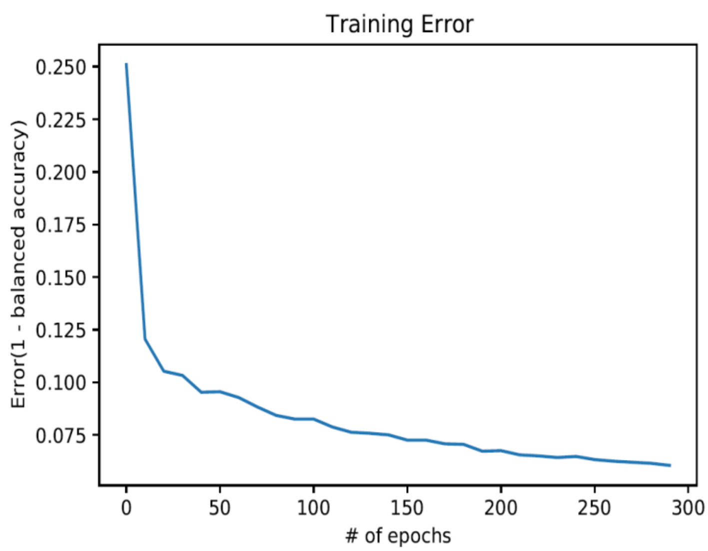

Figure 2.7 – training error for autoencoder w/ noise

2.3 Analysis of Results

The learning rate was determined by trial and error after running through all the epochs and the final value which gave the best results for this algorithm was *lr = 0.01*. We can see that the error rate keeps decreasing at a constant rate throughout the training and eventually for all of these classifiers it stays constant after the 250th epoch.

Figure 2.1, 2.2, 2.3, and 2.4 shows the comparison between the different confusion matrix for the regular classifier, autoencoder classifier, autoencoder with weights classifier, and the classifier with the SOFM weights. As expected, the digit 1 has the best performance throughout all these different types of classifiers. This is due to the fact that it is the easiest digit to denote as it is a straight line. As similar to the other classifiers, SOFM has bad performance on the digit 5 this is simply due to the fact that 5 has a lot of features and it might be hard to denote it.

3 Program Appendix

```
import pandas as pd
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import MultiLabelBinarizer
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
import numpy
from copy import deepcopy
from tqdm import tqdm

def train(X, lr=0.01, epochs=20000, verbose=False):
    Raw_Data_Shape = np.array([5000, 784])
    SOM_Network_Shape = np.array([12, 12])
    X_normalize = X / np.linalg.norm(X, axis=1).reshape(X.shape[0], 1)
    w = np.random.uniform(0, 1, (SOM_Network_Shape[0] * SOM_Network_Shape[1], X.shape[1]))
    w_normalize = w / np.linalg.norm(w, axis=1).reshape(SOM_Network_Shape[0] * SOM_Network_Shape[1], 1)
    network = np.mgrid[0:SOM_Network_Shape[0], 0:SOM_Network_Shape[1]].reshape(2, SOM_Network_Shape[0] * SOM_Network_Shape[1]).T

    lr_0 = lr
    lr_time_constant = 1000
    sig = np.max(SOM_Network_Shape) * 0.5
    sig_tau = 1000/np.log10(sig)
    w_current = deepcopy(w_normalize)
    lr = deepcopy(lr_0)
    sig = deepcopy(sig)

    for epoch in range(epochs):
        i = np.random.randint(0, X_normalize.shape[0])
        w_current = updating_weights(lr, sig, X_normalize[i], w_current, network)
        lr = decaying_lr(lr_0, epoch, lr_time_constant)
        sig = decaying_variance(sig, epoch, sig_tau)

        if verbose:
          if epoch % 1000 == 0:
            print('Epoch: {}; lr: {}; sigma: {}'.format(epoch, lr, sig))

    return w_current


def winning_neuron(x, Weight):
    return np.argmin(np.linalg.norm(x - Weight, axis=1))


def updating_weights(lr, var, x, Weight, network):
    k = winning_neuron(x, Weight)
    s = np.square(np.linalg.norm(network - network[k], axis=1))
    j = np.exp(-s/(2 * var * var))
    Weight = Weight + lr * j[:, np.newaxis] * (x - Weight)
    return Weight


def decaying_lr(lr_initial, epoch, time_const):
    return lr_initial * np.exp(-epoch/time_const)
```

```python
def decaying_variance(sig_initial, epoch, time_const):
    return sig_initial * np.exp(-epoch/time_const)

def receive_training_testing_set(training_file, testing_file):
    train = pd.read_csv(training_file, sep=",")
    y_train = train['label'].values.reshape(4000, 1)
    MLB = MultiLabelBinarizer()
    y_train = MLB.fit_transform(y_train)
    x_train = train.iloc[:, :-1].values

    test = pd.read_csv(testing_file, sep=",")
    y_test = test['label'].values.reshape(1000, 1)
    y_test = MLB.fit_transform(y_test)
    x_test = test.iloc[:, :-1].values

    return x_train, y_train, x_test, y_test

def splitting_data(data):
    df_train = pd.DataFrame()
    df_test = pd.DataFrame()
    for i in range(0, 10):
        df = data.loc[data['label'] == i]
        training_split = df.sample(frac=0.8, random_state=200)
        testing_split = df.drop(training_split.index)
        df_train = pd.concat([df_train, training_split])
        df_test = pd.concat([df_test, testing_split])

    df_train.to_csv('data/MNIST_Train.csv', sep=',', index=False)
    df_test.to_csv('data/MNIST_Test.csv', sep=',', index=False)

def convert_data(image_file, label_file):
    images = pd.read_csv(image_file, sep="\t", header=None)
    labels = pd.read_csv(label_file, header=None)
    images['label'] = labels

    return images


def winning_total(x_test, w):
    winning_total_dictionary = {}
    for i, j in enumerate(range(0, 1000, 100)):
        winning_total_dictionary[i] = []
        for xi in x_test[j:j + 100, ]:
            winning_total_dictionary[i].append(winning_neuron(xi, w))

    return winning_total_dictionary


def reformat(winning_neuron_dict):
        total_winning_dictionary = {}
        for digit in winning_neuron_dict:
            total_winning_dictionary[digit] = np.zeros(144)
            for ind in winning_neuron_dict[digit]:
                total_winning_dictionary[digit][ind] += 1
            total_winning_dictionary[digit] = total_winning_dictionary[digit].reshape(12, 12)
            total_winning_dictionary[digit] = total_winning_dictionary[digit] / 100
        return total_winning_dictionary
```

```python
def plot_winning_neurons(total_winning_dictionary):

    figs, ax = plt.subplots(2, 5)
    digit = 0
    for i in range(2):
        for j in range(5):
            ax[i][j].imshow(total_winning_dictionary[digit], cmap='inferno')
            ax[i][j].axis('off')
            digit+=1
    plt.show()


def plot_images(trained_weights):
    reshaped_w = trained_weights.reshape(12, 12, 784)
    figs, ax = plt.subplots(12, 12)
    for i in range(12):
        for j in range(12):
            ax[i][j].imshow(reshaped_w[i][j].reshape(28, 28).T, cmap='gray')
            ax[i][j].axis('off')
    plt.savefig('feature_map.pdf')


def plot_cm(y_true, y_pred, file_name):
    from sklearn.metrics import confusion_matrix
    cm = confusion_matrix(np.argmax(y_true, axis=1), np.argmax(y_pred, axis=1))
    import seaborn as sns
    df_cm = pd.DataFrame(cm, range(10), range(10))
    sns.heatmap(df_cm, annot=True, fmt='d', cmap ='OrRd')
    plt.savefig(file_name+'.pdf')
    plt.clf()


def main():
    # data = convert_data('MNISTnumImages5000_balanced.txt', 'MNISTnumLabels5000_balanced.txt')
    # splitting_data(data)
    # train = pd.read_csv('data/MNIST_Train.csv', sep=",")
    # test = pd.read_csv('data/MNIST_Test.csv', sep=",")
    # x_train, y_train, x_test, y_test = receive_training_testing_set('data/MNIST_Train.csv', 'data/MNIST_Test.csv')

    x_train, train_labels, x_test, test_labels = receive_training_testing_set('data/MNIST_Train.csv', 'data/MNIST_Test.csv')
    trained_weights = train(x_train, epochs=20000)

    total_winning_dictionary = reformat(winning_total(x_test, trained_weights))
    plot_winning_neurons(total_winning_dictionary)
    plot_images(trained_weights)


main()
```