# ECE/CS 559: Neural Networks Homework 4

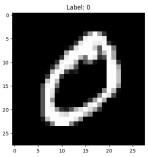
Samuele Pasquale

September 24, 2024

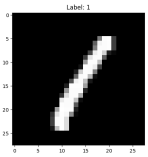
1. Question 1:

   (a) The code and the plots for the digits from 0 to 9 are shown below.

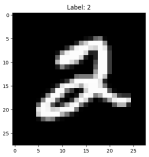
```python
def print_example_each_digit(Xraw, Yraw):
    """
    Function that shows one example of all the digits from 0 to 9
    :param Xraw: Xraw matrix
    :param Yraw: Yraw matrix
    :return:
    """
    # found flag
    found = 0

    # search up to digit 9
    while found < 10:
        # scan Yraw
        for i in range(len(Yraw)):
            # check if it is the target digit
            if Yraw[i] == found:
                # print the image
                print_image(Xraw[i], found)
                # update flag
                found += 1
                # exiting the for loop
                break


def print_image(row, number):
    """
    Function that shows the image of a given value
    :param row: row of Xraw, with 28x28 pixels
    :param number: Actual number shown in the image
    :return:
    """
    image = row.reshape(28, 28)  # MNIST images are 28x28 pixels
    plt.imshow(image, cmap='gray')
    plt.title(f"Label: {number}")
    plt.show()
```



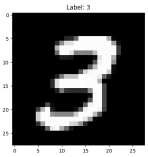

Number 0



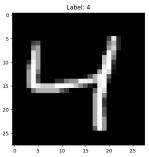

Number 1



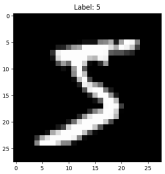

Number 2



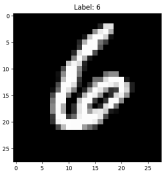

Number 3



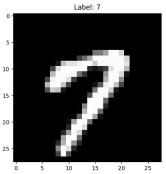

Number 4



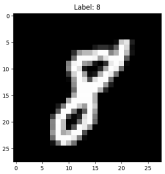

Number 5



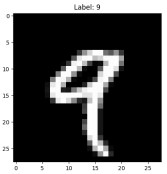

Number 6




Number 7




Number 8




Number 9

(b) The code is shown below.

```
    # set d to 10
    d = 10

    # generation of M, X, and Y
    M = generate_M(d, 784, 255*d)
    X = generate_X(M, Xraw)
    Y = generate_Y(10, Yraw)

    # Moore-Penrose pseudo inverse computation
    W = MoorePensorePseudoInverse(X, Y)
```

```
def generate_M(rows, cols, divisor, uniform_lower_bound=0, uniform_upper_bound
    =1):
    """
    Function that generates a rowsXcols matrix filled of random uniform values
        between the two
    given boundaries.
    :param rows: number of rows
    :param cols: numeber of columns
    :param divisor: divisor for M/M
    :param uniform_lower_bound: lower bound of uniform function
    :param uniform_upper_bound: upper bound of uniform function
    :return: return the generated matrix M
    """
    # generation of random matrix M
    M = np.random.uniform(uniform_lower_bound, uniform_upper_bound, (rows, cols)
        )
    # return matrix M
    return M / divisor


def generate_X(M, Xraw):
    """
    Function that generates X matrix as M * Xraw^T
    :param M: M matrix
    :param Xraw: Xraw matrix
    :return: matrix with dimension d * 70,000
    """
```

```
    # return  M * Xraw^T
    return M @ Xraw.T


def generate_Y(rows, Yraw):
    """
    Function that generates Y matrix, where each i-column represents
    the one-hot encoding of each i-row of Yraw
    :param rows: number of rows
    :param Yraw: Yraw matrix
    :return: Y matrix
    """
    # Y matrix zeroed out
    Y = np.zeros((rows, Yraw.shape[0]))
    # one-hot encoding
    Y[Yraw, np.arange(Yraw.shape[0])] = 1
    # return matrix Y
    return Y


def MoorePensorePseudoInverse(X, Y):
    """
    Function that computes the moore pensore pseudo-inverse given X and Y
        matrices
    :param X: X matrix
    :param Y: Y matrix
    :return: W = Y X^T (X X^T)^(-1)
    """
    # pseudo inverse with linalg  library
    X_pseudo_inv = np.linalg.pinv(X)
    # weights computation
    W = Y @ X_pseudo_inv
    # return weights
    return W
```

(c) The number of errors and the MSE of the predictor are reported below. As can be noticed from the table, increasing the parameter d causes the number of errors to decrease up to a certain value, 200, beyond which the number of errors saturates at around 10,000. Therefore, increasing d beyond 200 doesn't result in a significant improvement in the number of errors. In conclusion, d=200 is a good choice, while a higher value doesn't further decrease the error rate of the predictor.

Finally, it can be noted that the error probability, considering 36,522 errors out of 70,000, is 52.17 %. On the other hand, if the classification were to occur randomly, the error probability would be 9 out of 10 for each image, resulting in a probability of 90 %. Therefore, it can be concluded that the obtained model is still more accurate than a random choice.

| $d$ | **MSE** | **Number of Errors** |
|---|---|---|
| 10 | 0.7645 | 36,522 |
| 50 | 0.5372 | 15,856 |
| 100 | 0.4649 | 12,467 |
| 200 | 0.4221 | 10,901 |
| 500 | 0.3940 | 10,399 |

(d) In the table 1, the number of errors is shown. As can be noticed the number of errors is higher than the number of errors obtained in (c) with d = 100.

The approach taken to improve the accuracy of the algorithm involved training on different numbers of epochs and for different $\eta$, with the results shown in Figure 6 and Table 2. From the results obtained, it is concluded that a smaller $\eta$ allows for slower convergence with fewer oscillations towards a local minimum. However, as shown in the graphs, while the MSE for $\eta = 0.001$ and $\eta = 0.005$ are similar, if $\eta$ is reduced too much, the updates during training will be too small, and thus for a fixed number of epochs, the model might not escape one minimum to reach a better one. Therefore, in such cases, a larger $\eta$ would allow for larger steps towards obtaining the optimal solution.

Finally, the impact of the number of epochs on the final result is analyzed. It is observed that, given an optimal $\eta$, as described previously, the number of epochs has relatively little impact, and for all tests, the MSE remains of the same order of magnitude. Thus, an excessive number of epochs not only affects the execution time, making it unnecessary for the learning phase, but could even cause an error known as overfitting (source: https://medium.com/). In this case, the model becomes excessively adapted to the training data, subsequently losing accuracy on the test data.

| $d$ | **Number of Errors** |
|---|---|
| 100 | 13918 |

Table 1: Number of errors for different $d$ values

| Errors | Epochs | Learning Rate (eta) |
|---|---|---|
| 25524 | 10 | 0.0001 |
| 17802 | 10 | 0.001 |
| 14541 | 10 | 0.005 |
| 22990 | 20 | 0.0001 |
| 15670 | 20 | 0.001 |
| 12860 | 20 | 0.005 |
| 20260 | 30 | 0.0001 |
| 15262 | 30 | 0.001 |
| 12973 | 30 | 0.005 |
| 20404 | 40 | 0.0001 |
| 14082 | 40 | 0.001 |
| 12271 | 40 | 0.005 |
| 18896 | 50 | 0.0001 |
| 13783 | 50 | 0.001 |
| 12465 | 50 | 0.005 |

Table 2: Comparison of Errors, Epochs, and Learning Rate (eta)

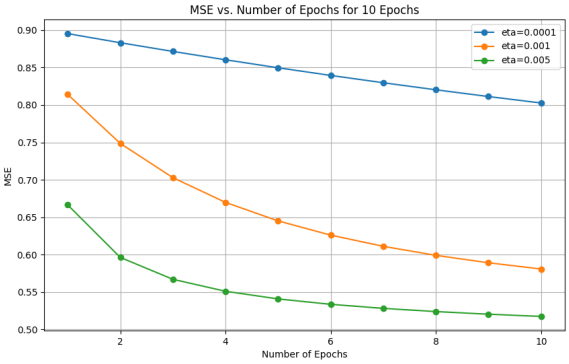



Figure 1: 10 epochs

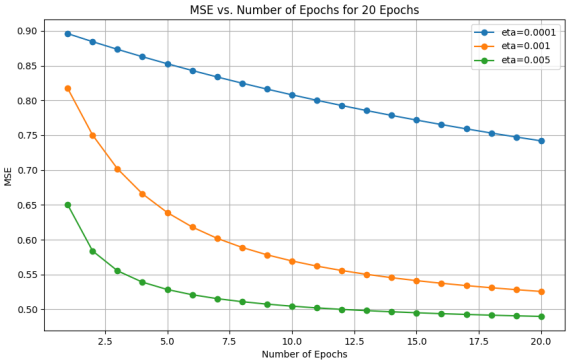



Figure 2: 20 epochs

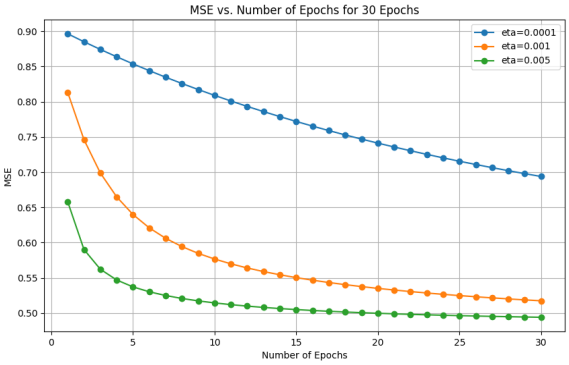



Figure 3: 30 epochs

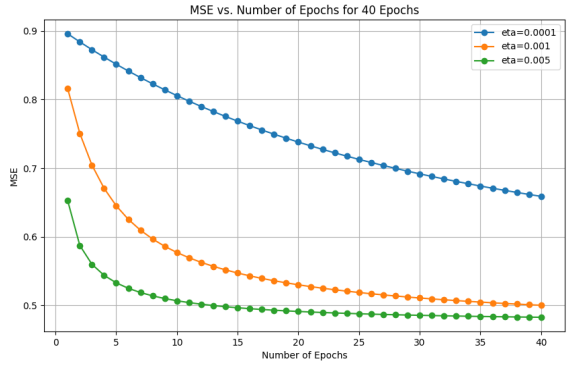



Figure 4: 40 epochs

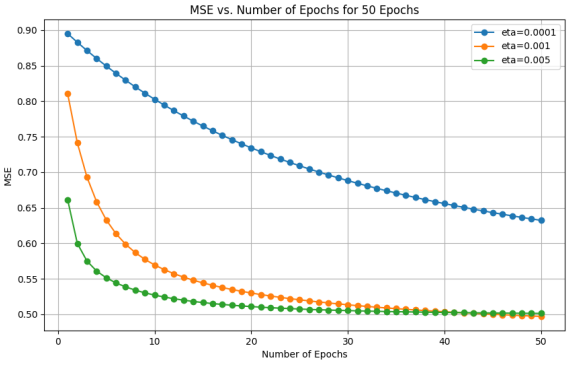
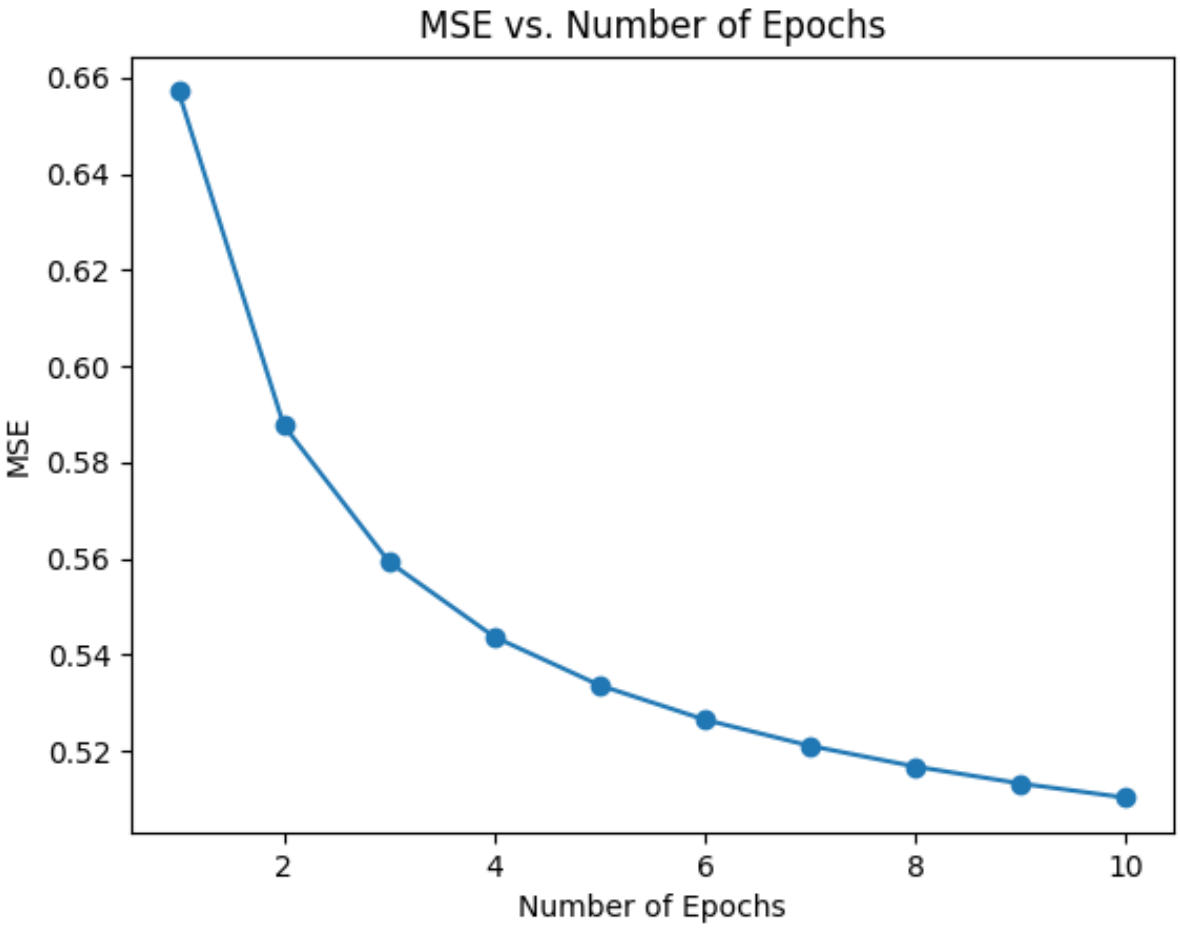



Figure 5: 50 epochs

Figure 6: Tentative of algorithm improvements


MSE vs. Number of Epochs
MSE
0.66
0.64
0.62
0.60
0.58
0.56
0.54
0.52
2
4
6
8
10
Number of Epochs