# CISC 452 Assignment 2

Aaron Safer-Rosenthal | 20068164 | 17asr

## Model 1

### Training

#### Confusion Matrix

| Output \ Target | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 863 | 0 | 54 | 9 | 6 | 31 | 12 | 4 | 58 | 8 |
| 1 | 0 | 1073 | 75 | 10 | 10 | 9 | 5 | 5 | 30 | 9 |
| 2 | 12 | 10 | 735 | 58 | 8 | 19 | 23 | 33 | 14 | 5 |
| 3 | 4 | 3 | 75 | 781 | 1 | 49 | 0 | 7 | 19 | 7 |
| 4 | 3 | 2 | 8 | 1 | 765 | 29 | 8 | 14 | 17 | 75 |
| 5 | 34 | 2 | 7 | 107 | 6 | 633 | 27 | 1 | 128 | 17 |
| 6 | 53 | 2 | 30 | 2 | 23 | 18 | 861 | 1 | 26 | 6 |
| 7 | 7 | 0 | 24 | 11 | 3 | 8 | 2 | 891 | 13 | 31 |
| 8 | 4 | 41 | 12 | 9 | 5 | 59 | 20 | 1 | 626 | 8 |
| 9 | 0 | 2 | 12 | 22 | 155 | 37 | 0 | 51 | 43 | 843 |

Precision: 82.23%

Recall: 83.52%

### Testing

#### Confusion Matrix

| Output \ Target | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 869 | 0 | 11 | 52 | 0 | 104 | 13 | 16 | 21 | 10 |
| 1 | 0 | 1050 | 1 | 2 | 3 | 12 | 3 | 14 | 3 | 6 |
| 2 | 3 | 42 | 786 | 68 | 7 | 17 | 21 | 42 | 54 | 10 |
| 3 | 18 | 33 | 114 | 757 | 1 | 169 | 1 | 38 | 296 | 18 |
| 4 | 0 | 0 | 25 | 3 | 624 | 2 | 69 | 17 | 3 | 60 |
| 5 | 58 | 1 | 2 | 43 | 22 | 480 | 35 | 5 | 49 | 15 |
| 6 | 25 | 2 | 60 | 21 | 19 | 32 | 813 | 0 | 17 | 1 |
| 7 | 2 | 2 | 4 | 54 | 6 | 4 | 0 | 859 | 14 | 93 |
| 8 | 3 | 4 | 3 | 5 | 3 | 60 | 2 | 1 | 505 | 5 |
| 9 | 2 | 1 | 26 | 6 | 297 | 12 | 1 | 36 | 12 | 791 |

Precision: 77.5%

Recall: 80.9%

### Critical Discussion

First, I will describe how I chose my parameter values. I initially chose to use 10 input layers, because there were 10 different possible values. However, I quickly realized that this was not the right decision and that there should be 10 output nodes, one for each potential value. After this, I realized to use 784 input layers (28*28), because that was the amount of values that each input would have. Next, I needed to decide how many hidden nodes to use. The assignment said to use only one hidden layer, so I did. As for the amount of nodes, I settled on 30. Because of how long it took to train the network, I was not able to test many different values for the number of hidden nodes. In deciding 30 hidden nodes, I used a rule of thumb, Nh = Ns / (α∗(Ni+No)), where Ni = number of input neurons; No = number of output neurons; Ns = number of samples in training data set; α = an arbitrary scaling factor usually 2-10. This lead to Nh = 60000/(2*(784+10)) = 25. I also used this formula with α=3, which = 37. I averaged these to 30 and trained the program with 30 hidden nodes. Satisfied with the result that it gave, I kept 30 hidden nodes. For number of epochs, I settled on 300. I found that the network was not getting much more accurate after 300 epochs, and given the long training times, I did not increase it. For choosing the learning rate, I tested on three different values, 0.01, 0.5 and 1. Since the learning rate is frequently determined by trial and error, I would have ideally tested training with more learning rates; however, once again, given the long training times, I only tested with these three values. I found the most efficient training with 0.5, so I stuck with it.

The last parameter choice I needed to make was for the mini batch size. This ties into my model design choices, so I will first describe why I decided to use mini batches. Mini batch training involves using a set number of training inputs, which is (substantially) less than the total number of training inputs. The point of this is to speed up training, since using the entire batch of training inputs on each epoch would take a long time to train. I chose to use mini batches with 10 training inputs per batch. I found that this would give almost identical results to using the entire batch of data, but in much less time.

The main limiting factor when designing the network was the time it took to train it. This did not let me play around with many different parameter values, which would have been ideal, to select the best ones. However, I am confident that I chose good ones anyways. It is cool that we are writing programs that are computationally intensive, and it shows why companies need supercomputers to perform their machine learning.

## Model 2A

### Training

#### Confusion Matrix

| | | *Target* | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| *Output* | 0 | 5680 | 1 | 11 | 0 | 1 | 5 | 6 | 5 | 2 | 8 |
| | 1 | 0 | 6606 | 28 | 4 | 17 | 3 | 2 | 12 | 49 | 7 |
| | 2 | 4 | 17 | 5069 | 38 | 3 | 14 | 1 | 70 | 29 | 0 |
| | 3 | 8 | 42 | 583 | 5872 | 16 | 164 | 1 | 203 | 115 | 52 |
| | 4 | 16 | 7 | 53 | 0 | 5454 | 3 | 4 | 21 | 1 | 76 |
| | 5 | 9 | 9 | 6 | 117 | 14 | 5014 | 67 | 4 | 92 | 123 |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 6 | 61 | 1 | 35 | 3 | 69 | 75 | 5793 | 3 | 27 | 0 |
| 7 | 1 | 18 | 22 | 19 | 20 | 4 | 0 | 5902 | 1 | 78 |
| 8 | 143 | 37 | 149 | 54 | 42 | 129 | 44 | 11 | 5525 | 51 |
| 9 | 1 | 4 | 2 | 24 | 206 | 10 | 0 | 34 | 10 | 5554 |

Precision: 94.15%

Recall: 94.15%

## Testing

### Confusion Matrix

| | | *Target* | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| *Output* | 0 | 946 | 0 | 1 | 1 | 1 | 2 | 5 | 2 | 3 | 2 |
| | 1 | 0 | 1118 | 5 | 1 | 0 | 1 | 2 | 5 | 9 | 6 |
| | 2 | 2 | 0 | 861 | 1 | 2 | 3 | 2 | 29 | 9 | 0 |
| | 3 | 4 | 6 | 105 | 962 | 2 | 36 | 1 | 43 | 24 | 13 |
| | 4 | 1 | 0 | 12 | 1 | 903 | 2 | 5 | 4 | 6 | 24 |
| | 5 | 2 | 2 | 1 | 26 | 3 | 802 | 11 | 2 | 20 | 17 |
| | 6 | 15 | 3 | 9 | 0 | 20 | 17 | 918 | 0 | 11 | 0 |
| | 7 | 1 | 1 | 5 | 5 | 3 | 1 | 0 | 933 | 3 | 10 |
| | 8 | 8 | 5 | 31 | 9 | 7 | 24 | 14 | 5 | 883 | 13 |
| | 9 | 1 | 0 | 2 | 4 | 41 | 4 | 0 | 8 | 6 | 924 |

Precision: 92.5%

Recall: 92.5%

## Critical Discussion

For this model, I used scikit-learn's (sklearn) MLP Classifier model, by importing it into python 3. For the parameter values that it shared with model 1 (learning rate, number of hidden nodes, etc) I used the same values that I used in model 1, as described in the assignment description. Because of this, using the model was mainly a plug-and-play process, after which I compared the model's results with my model 1. The sklearn model performed much better than my model, being 15% more accurate, while achieving this in a much shorter period of time. Outside of the sklearn model probably being optimized better than my model 1, it also was able to identify if the training did not improve over 10 consecutive epochs. I was able to see this, because the model had an option which allowed it to print it's progress over each epoch. If I had to remake my model 1, I would probably include a feature that would allow it to identify if it is not becoming more accurate over time. This would allow the program to end quicker and save computing time.

# Model 2B

## Training

### Confusion Matrix

| Output \ Target | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 5923 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 6742 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 5958 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 6131 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 5842 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 5421 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 5918 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 6265 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5851 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5949 |

Precision: 100%

Recall: 100%

## Testing

### Confusion Matrix

| Output \ Target | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 969 | 0 | 3 | 0 | 2 | 2 | 4 | 1 | 5 | 0 |
| 1 | 0 | 1124 | 1 | 0 | 0 | 0 | 2 | 3 | 1 | 2 |
| 2 | 1 | 4 | 1008 | 4 | 3 | 0 | 2 | 5 | 3 | 0 |
| 3 | 0 | 1 | 1 | 994 | 1 | 7 | 1 | 3 | 3 | 2 |
| 4 | 0 | 0 | 2 | 0 | 962 | 1 | 4 | 0 | 3 | 7 |
| 5 | 0 | 1 | 0 | 4 | 0 | 871 | 3 | 0 | 3 | 3 |
| 6 | 4 | 2 | 2 | 0 | 2 | 4 | 941 | 0 | 2 | 1 |
| 7 | 2 | 1 | 6 | 2 | 2 | 1 | 0 | 1005 | 2 | 3 |
| 8 | 3 | 2 | 9 | 3 | 0 | 4 | 1 | 4 | 950 | 4 |
| 9 | 1 | 0 | 0 | 3 | 10 | 2 | 0 | 7 | 2 | 987 |

Precision: 98.11%

Recall: 98.11%

## Critical Discussion

The goal of this model was to improve on the results from model 2 (and model 1, I guess) by choosing different parameter values. The values I changed were increasing the number of hidden nodes to 200 and decreasing the learning rate to 0.05. I changed the values one at a time, starting with increasing the hidden nodes. I first increased from 30-50, then from 50-100, from 100-200 and from 200-300. I got the

most accurate results from using 200 hidden nodes, so I stuck with it. Once I moved to the learning rate, I tried 0.05 and 0.01. The 0.01 training took a long time and the 0.05 gave good results. Satisfied, I chose 200 hidden nodes and 0.05 learning rate as my new parameter values. In retrospect, I could have tried model 1 with 200 hidden nodes but going back and changing it would defeat the purpose of trying to make model 2b better than it. However, for future models, I will not be afraid of using many hidden nodes, as it can be beneficial.