**Ryan Connolly**

**Problem 1**

**System Description**

I chose to give my neural network 196 hidden neurons for two reasons: 1) it performed much better with 200 neurons than 100 neurons (my initial tests), and 2) 196 is a perfect square number, making plotting heatmaps of the output of the 196 hidden neurons for visualization purposes very easy.

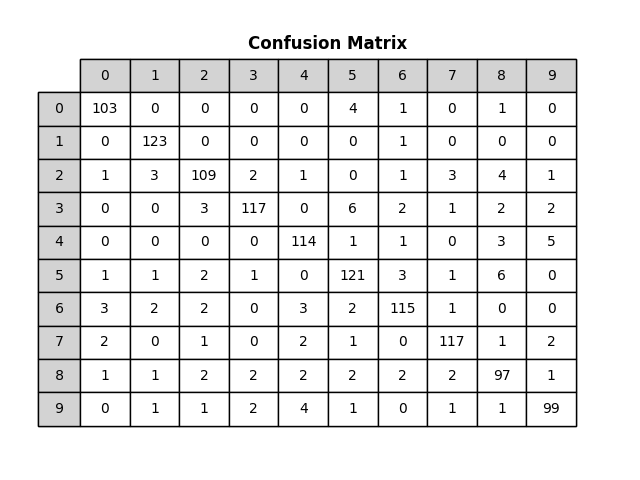
For the learning rate, I chose 0.001. First, I tried larger values, like 0.1 and 0.01, but with the addition of momentum to the gradient descent training, my network was prone to drastic fluctuations of weights. To counteract this, I chose a smaller learning rate, 0.001. As mentioned, the momentum caused instability in the values, so I gradually lowered my α value for momentum calculation from 0.9 to 0.3. This prevented weights from collectively being too close to 1.0 and exceeding it, causing wildly large magnitudes for the output neurons’ values.

To generate my initial weights, I used the method of Gaussian distribution with a mean of 0 and a standard deviation of the square root of 2 / *N*, where *N* is the number of previous-layer neurons. This means that for the weights *wjk* (input-to-hidden), *N* was equal to 784, and for the weights *wij* (hidden-to-output), *N* was equal to 196 (as according to my final hidden neuron count discussed above).

I used *f(s)* = tanh(*s*) for my activation function, as its horizontal asymptotes of *y* = -1, 1 keep the output within that desired range. My output thresholds were 0.25 and 0.75, to stay even and spaced out. The criteria for which I stopped training was when the training error was less than or equal to 0.1.

**Results**

|  |  |
| --- | --- |
| **Figure 1.1: Training Set Confusion Matrix** | **Figure 1.2: Test Set Confusion Matrix** |

 A picture containing table

Description automatically generated

|  |  |
| --- | --- |
| **Figure 1.1:** confusion matrix for the training set’s output from the neural network. Rows represent actual number in the image, columns represent number classified by the network. | **Figure 1.2:** confusion matrix for the test set’s output, after tuning of weights through training set. Rows represent actual number in the image, columns represent number classified by the network. |

0 1 2 3 4 5 6 7 8 9

.055 .008 .128 .120 .081 .110 .102 .071 .142 .100

|  |  |
| --- | --- |
| **Figure 1.3: Training Set Error Fraction** | **Figure 1.4: Test Set Error Fraction** |

Chart

Description automatically generatedChart

Description automatically generated

|  |  |
| --- | --- |
| **Figure 1.3:** training set’s error fraction (percentage of incorrect output) plotted as a function of epoch iterations (1-41). | **Figure 1.4:** test set’s error fraction plotted as a function of epoch iterations (1-41), with trained weights. |

**Analysis of Results**

Figure 1.1 depicts “confusion matrix,” or the classification results of the trained network upon the training set, which is shown to be proficient. The diagonal, which contains the counts of correctly classified images for each number, has all but two cells at least 100 total, and those two are quite close at 97 and 99. Meanwhile, the rest of the cells all have values in the single digits, so it can be concluded that the network is generally accurate. It is especially accurate regarding 1s and 0s, only mislabeling true 1s and 0s once and six times, respectively, with 7 and 4 close behind as nine and ten times. This was largely expected, as 1 and 0 have simple shapes compared to others like 5 and 8. The number that was classified wrong the most according to mislabels divided by total occurrences is 8, marked wrongly 14.2% of the time, followed by 2f and 3 with 12.8% and 12.0% respectively. 8 was somewhat surprising, as while it is a complex shape, I also thought it is a distinct one. However, I conclude that its loops are easily confused with a part of every other number’s shape, since its mislabels are very evenly spread across every other number. 1 and 0 were marked wrongly 0.8% and 5.5% of the time.

Figure 1.2 depicts the confusion matrix for the trained network’s results upon the test set. This matrix shows the test set data to be less favorable than the trained network. First, it should be noted that there are exactly 100 total images of each digit read in the test set, so each cell value can also be read as a percentage (a cell value of 77 also signifies 77%). There is a double-digit number in a wrong-classification cell, showing that fifteen 4s were mistaken for 9s. This is not surprising, as 9 was the most common mistake made regarding 4s in the training set as well, and 4 and 9 have remarkably similar shapes, especially when 4 is written with a closed, triangular top half (as opposed to the open-on-the-top, square top half 4 that is often used in handwriting).

Figures 1.3 and 1.4 show the error fraction of the training and test sets, which is the total wrongly classified inputs divided by the total inputs amount, over epoch iterations. It took 41 epochs to reach my training error fraction goal of ≤ 0.1, reaching 0.0913. The test set’s error fraction is the same no matter how many epochs it runs through, as it is not training for this set, and it came out to be 0.1490, which is higher than my threshold. This is reflective of the same behavior seen in Figure 1.2 and discussed above. However, there is logic behind this, as it makes sense that the network would be better and less erroneous at classifying its training set than other sets, as the training set is the one that it has based all its opinions, in the form of the weights, upon.

**Problem 2**

**System Description**

My network for Problem 2 also had 196 neurons, as we were instructed to use the same amount. Continuing this theme, I ended up using largely the same hyperparameters as Problem 1. I found that it worked well with the number of neurons and the balance between them seemed to be maintained with the change of purpose/application. However, I did modify the learning rate and the momentum coefficient to be 0.03 and 0.8 respectively. This is an increase for both from Problem 1, which I did with the intention to increase contrast in the outputs.

**Results**

**Figure 2.3: J2 vs Epochs**

**Chart, line chart

Description automatically generated**

**Figure 2.3:** J2, modified for reconstruction,   
as a function of epochs iteration.

**Features**

**Figure 2.4: Hidden Neuron Feature Heatmaps**

**A picture containing table

Description automatically generated**

**Figure 2.4:** heatmaps of weights *wjk*for 20 randomly selected hidden neurons.

**Sample Outputs**

**Figure 2.5: Original and Reconstructed Image Comparison**

**Table

Description automatically generated**

**Figure 2.5:** 8 random samples of an original input image and the reconstructed output image.

**Analysis of Results**

Clearly my results are not finished, but I do see the trend here. The network tunes its weights through the header neurons to reflect the input images’ pixels/neurons but condensed, and then when the weights *wij* are used to expand, it comes back.