ANN- Feed Forward Neural Network with Backpropagation

Assignment 3

October 12, 2020

Course: Fundamentals of Artificial Intelligence

Course Code: 5dv124

Group: Reza Firouzi, mrc20rfi
Derek Yadgaroff, mai20dyf

Summary

We have trained 4 perceptrons to classify the digits 4, 7, 8 and 9 from a subset of the MNIST handwritten digit dataset. Each perceptron is a binary classifier that represents a specific digit. For example, if the perceptron detecting the digit “4” was a human, it would think to itself “yes this digit is a 4” or “no this digit is not a 4”.

Some interesting things about our solution:

- It uses a matrix implementation to train all 4 perceptrons at once
- It uses the sigmoid activation function (cleared by multiple TA’s)
- It uses a one-hot encoder
- It normalizes the image data
- It executes on a 2013 Mac Air in 9 seconds

Input-Output Nodes

There is a single layer of input nodes which represent each pixel of an image. Therefore there are 784 input nodes. This number comes from the flattening of the actual image which is 28px x 28px (28*28=784).

These input nodes also output their value to each perceptron. Therefore there are 4 outputs per node, for a total of 3136 connections.

Weights

The output nodes’ connections are weighted. We therefore implement the connections through a weights matrix. The weights matrix is 784 rows by 4 columns. This means that the rows represent the 784 weights which match each of the pixel nodes. The 4 columns represent the 4 perceptrons.

When we take the dot product of the weights and the input nodes, we actually use the transpose of the weights matrix. This is because the weights matrix is organized by input nodes as the rows, and perceptrons on the columns. What we really need though is to organize this from the perspective of each perceptron. Each perceptron is concerned only with it’s incoming weighted inputs.

Perceptrons

Each perceptron receives the dot product of the incoming weights multiplied by the corresponding pixel value. This scalar value is then put through an activation function, in our case, the sigmoid function. The sigmoid essentially outputs a 0 or 1. In the future

we would have liked to implement RELU as this seems to be a more optimal implementation.

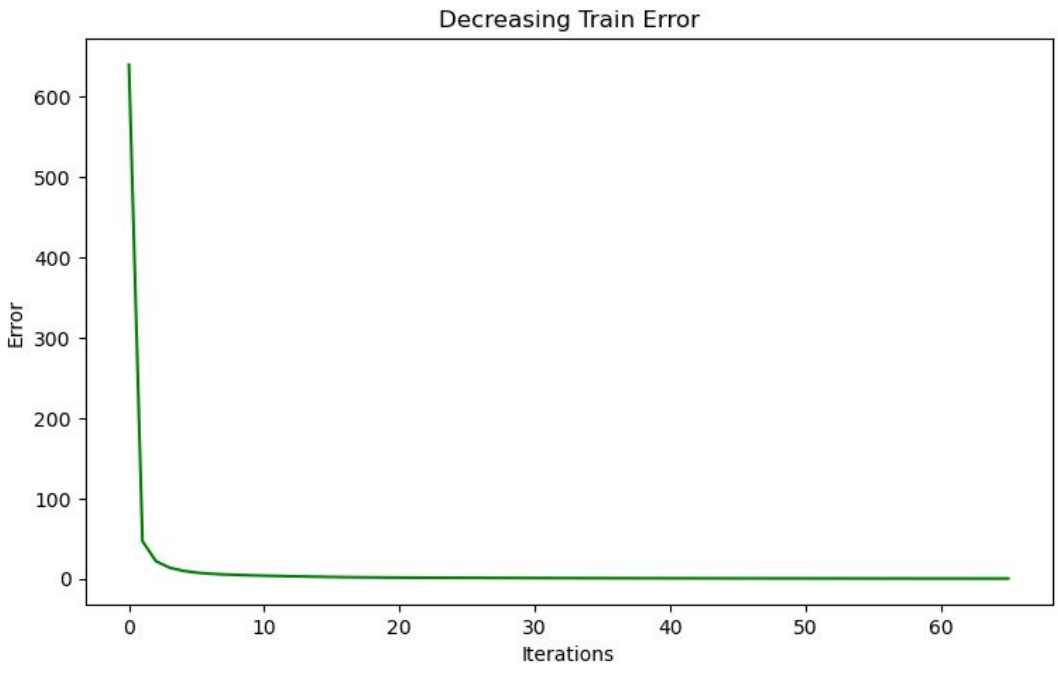
Updating the Weights

Once the perceptron is activated or not, the backpropagation begins, although that term is typically reserved for more complex neural networks to our understanding. This means that we calculate an error based on whether the digit's activation was correct or now, and how badly incorrect it was. This output value then gets added to the weights matrix. At some point the updates to the weights matrix becomes negligible so we stop

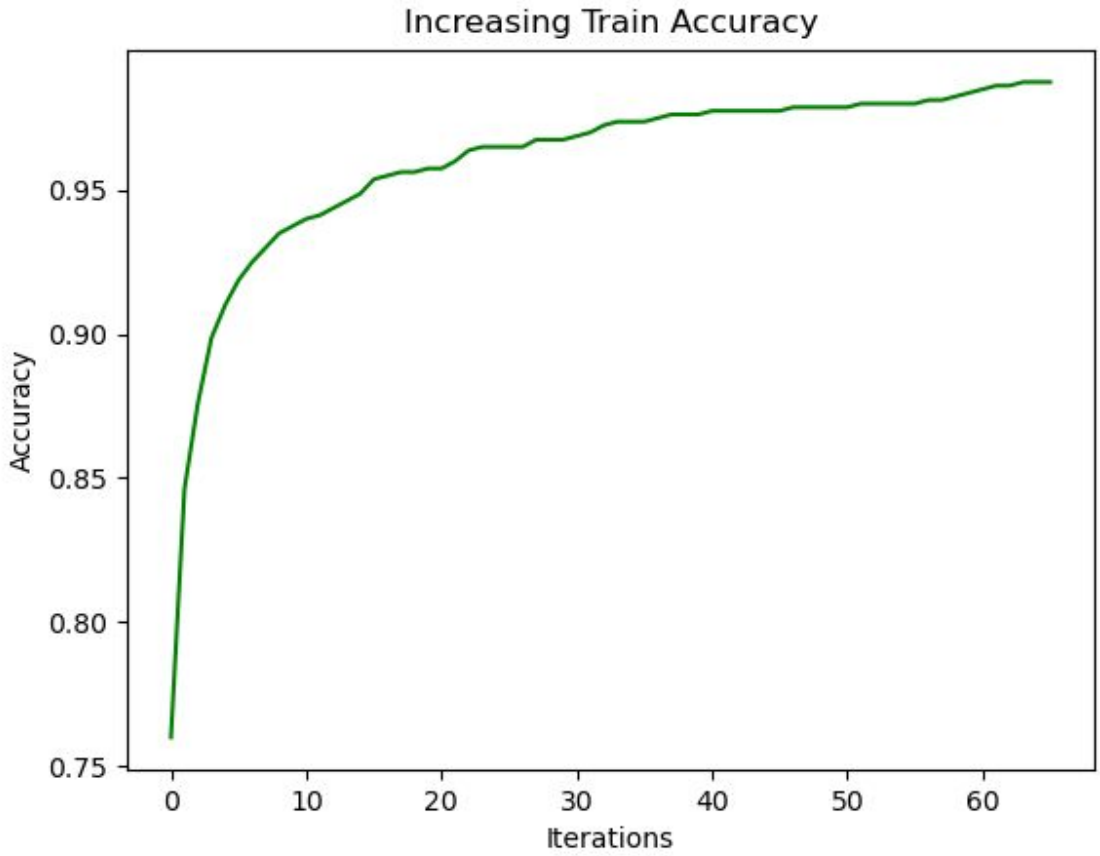
Overfitting

We implemented a basic form of early stopping which helps prevent overfitting and also saves on computing time. In this case, when we detect that updates to the weights are no longer yielding a significant reduction of error, we terminate the training loop.

Plot of Error



Plots of Accuracy



Collaboration

Reza and Derek are a very good team and continue to do peer programming virtually and at the library.

We had no significant problems as we are both familiar with numpy and basic neural networks.

We are looking forward to more advanced programming assignments in the future.