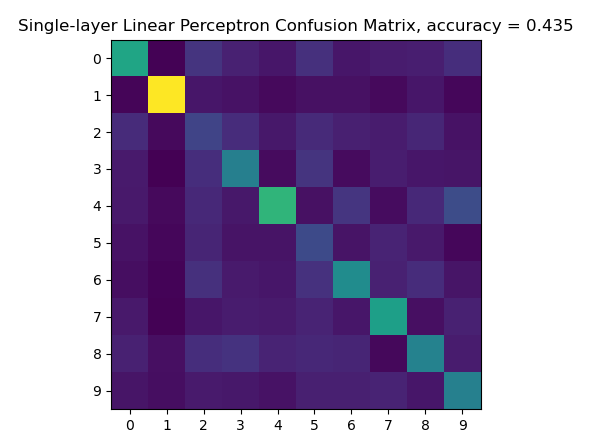
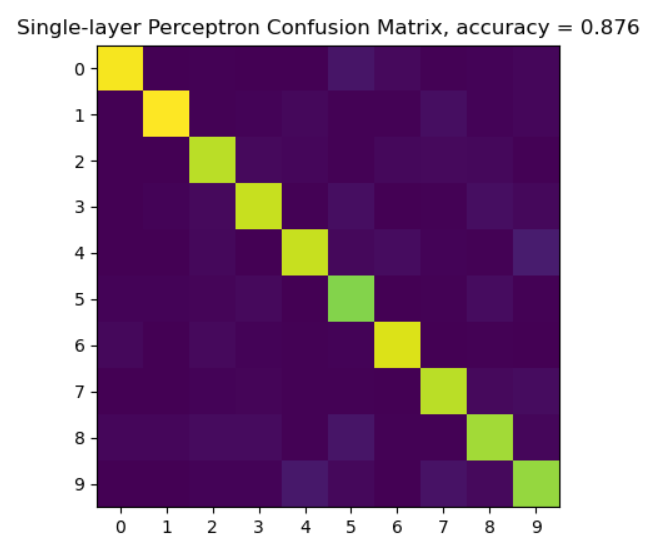
CSCI5561 HW4

Anton King

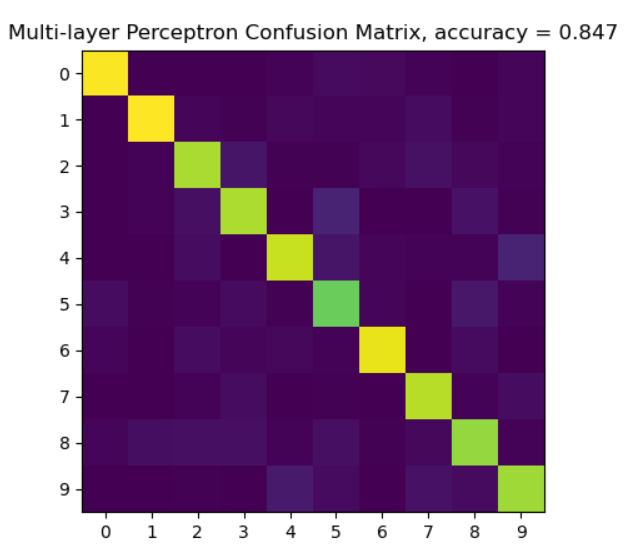
November 29 2021



1. Single layer Linear perceptron. This method was implemented using a single layer fully connected, where inputs of the MNIST digits dataset connected directly to a 10 node output, directly mapping to the prediction of the digit. Since this is very simple model, we see that only very simple digit is well predicted- the digit 1. Other than that, very low accuracy is recorded.



1. Single layer perceptron. This method utilized a softmax output function to clamp the output to a probability between 0 and 1. The inputs are connected to a fully connected layer, and the output from the fully connected layer is fed into the cross entropy softmax which produces a class probability. As shown in the figure to the right, the accuracy is much higher than the single linear perceptron, and it is able to accurately predict almost every digit in the dataset.



1. Multi Layer perceptron. This method utilizes a single hidden layer with a ReLU activation function. The inputs are fed into the ReLU function with outputs the input if it is positive, and the input divided by 100 if it is negative (Leaky ReLU implementation). This output is fed into a fully connected layer, which is then fed into a softmax to produce a class probability. Due to a problem with my ReLU backward propagation function, this implementation actually does worse than the single layer perceptron. My ReLU backward propagation function does not take into account dl\_dy correctly, and due to this it does not change the input x\_i when x\_i is positive. It should set x\_i to dl\_dy when x\_i is positive, however due to unknown reasons this produced an error rate of around 40% so clearly there is an issue somewhere else. Code is provided in cnn.py
2. Convolutional neural network. I was not able to successfully train a convolutional neural network. I have working functions for all but flattening\_backward() and train\_cnn(), however in my .ipynb included with the submission, there is what i believe to be a very close attempt at train\_cnn. In theory, the input photo is fed into a convolutional layer where feature extraction occurs using the convolutional operation. This is working in my code. Then, it is fed into the ReLU function from earlier in my code. The output from ReLU is fed into a 2x2 max pooling function with halves the dimensionality of the data. This is working in my code. The output is vectorized, fed into a fully connected layer, and ffed into a softmax so a class prediction can be made. I think my code is very close, I simply ran out of time to debug the dimensionality errors. Please look over cnn.py and the ipy notebook as there is working code for cnn there, just not the final training