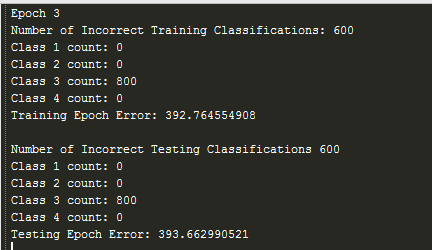
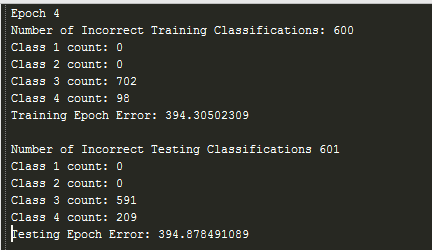
# Back propagation

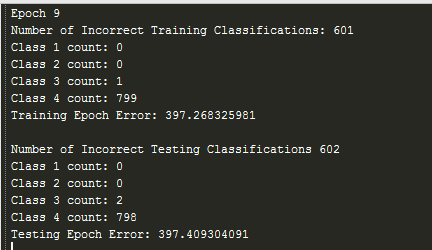
The code I am providing is written in Python. It uses the scipy library for the exp() function. All activation functions for the network were assigned as the logistic function with a slope parameter equal to 1. Unfortunately, there is a bug in my code that I have been unable to track down over the last couple days which has made accumulating results impossible.

For some reason, the network always has a higher output on one output node no matter what class an input presentation belongs too. This leads to a classification of all points as a single class. Even though the errors are calculated so it would lower the weights for an incorrect classification and raise the weights for a correct classification, it all seems to converge to a random class.

For example, a network with two hidden layers of five nodes each at epoch 3:

One epoch later, we begin converging to class 4:





A couple of epochs later, it fully converges and classifies everything as class 4. There are no improvements made and the epoch error stays around 400 (meaning the local gradient is negative in the nodes that incorrectly classified it). The classifications that occur are based off a winner take all strategy for the output nodes. As far as I can tell, all forward propagation is correct and all backward propagation corrects weights in the right direction. I’m at a loss at this point as to why I have this convergence.

I have checked the network structure and the connections are correct, only connecting one layer to the next, there are no connections that skip layers, nor are any input values pushed backward through the network.

My code can be found at <https://github.com/teberger/neural_networks/tree/master/homework5>. Any input on what is going on would be appreciated.