Multi-Layer Perceptrons

For this project I used the sigmoid activation function with one hidden layer of three nodes and one output node. This was large enough for me to get a 99.9% accuracy for the 0-1 test set. I have the amount of nodes variable so that it can be easily changed.

Accuracy over three tries: Text

Description automatically generated

\*\*\*Explains the code (not necessary but a little extra info on what I did if my code isn’t clear)

For this project I first created a neural Net class to hold all the lists I have so I didn’t need to pass a bunch of lists to every function. Next I separated my code out into many functions so it is easy to follow. I first open a file then create the random weights and run the run\_updates function on every line in the training set. The run\_updates calls the forward pass to get the output of the network (it also returns the h\_in, h\_out, and raw\_out since those are needed for the back\_pass). Then I run the back\_pass function from the run\_updates function which calculates the deltas and updates the weights. Lastly, I have a test function that takes inputs and runs them through the neural net and gives back an output which I compare to the answer and get a percentage correct.

End code explanation\*\*\*

For the bonus part of this projects I also used the sigmoid activation function with one hidden layer and one output node, but I used 15 hidden nodes at that hidden layer. I had a little difficulty with this one however, because the setup of my neural net keeps the output between 0 and 1. To get this to work, I divided the answer to the outputs by 4 to keep the numbers between 0 and 1. This was updated right when I read in the data, so it occurs for error as well. This made me also update my testing function to see if the expected output was between 0 and .125 for the real answer\*4 to be 0. The rest of the code was the same.

Accuracy over three tries:

Text

Description automatically generated