Ryan Wendling
Final Assignment
Zhang Fall 2016
Discussion Paper

The previous problem of assignment 3 was to take a large and complex dataset and be able to somewhat accurately predict a feature of an item within that dataset using an artificial neural network with back-propagation. More specifically, we sifted through 22 characteristics of over 6000 mushrooms via this “ANN-BP” in an effort to determine the edibility/toxicity of the next 2000.

Since then, our goal was to improve the accuracy and speed of our ANN from assignment 3. In this newer iteration of our ANN, we have added an additional hidden layer, more than doubled our number of weights, normalized our input parameters and added a “momentum” function to the update weight logic.

The added hidden layer and additional weights are pretty self explanatory. Likewise, the implementation was essentially copying/ expanding on some of the code we had already written. I read that more weights reduce the chance of “underfitting”. Underfitting occurs when there are too few weights in the hidden layers to adequately detect the signals in a complicated data set.

Standardizing the inputs took a bit more work. Originally, the inputs for the mushroom’s characteristics were directly related off their equivalent ASCII value. So the mushroom characteristic value of ‘?’ would be read as ‘63’, while the characteristic value ‘p’ would be ‘112’. To normalize these inputs, I looked at all the potential values for each characteristic and instead assigned them from 1 to however many different values there were for that characteristic. I read that doing this helps increase training speed and lessens the chance of falling into a local optima.

Finally, momentum adds the previous change in a weight multiplied by the momentum constant to the current weight we are updating. This momentum helps us from converging to a local minimum while also increasing our overall speed of convergence of the system. Momentum is like a rock rolling downhill, gaining speed and flying over small divots until it reaches the bottom.

With these changes, the overall speed of the program is a tad slower, due to more computations being performed. The accuracy however has certainly increased. Originally, the accuracy rating would hover between the upper 3000s to lower 4000s(out of 6909 attempts). The accuracy now ranges between the mid 4000s to upper 5000s! Reading online, there are a lot of advanced algorithms to make the whole process even more efficient. Some like adaptive learning rates and concurrency seemed doable, but others like “parallel particle swarm optimization” seemed very complex and math intensive. I imagine that it would be strategies like that to achieve high accuracy without sacrificing run time.