

Adam Gincel

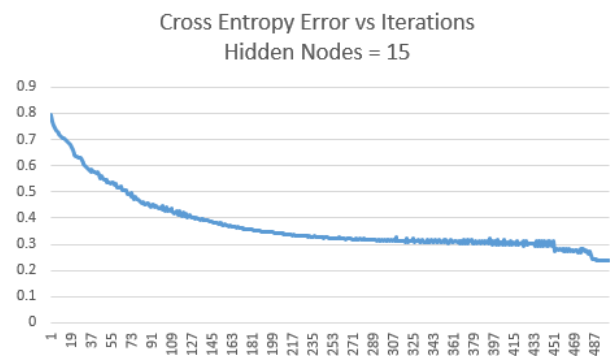
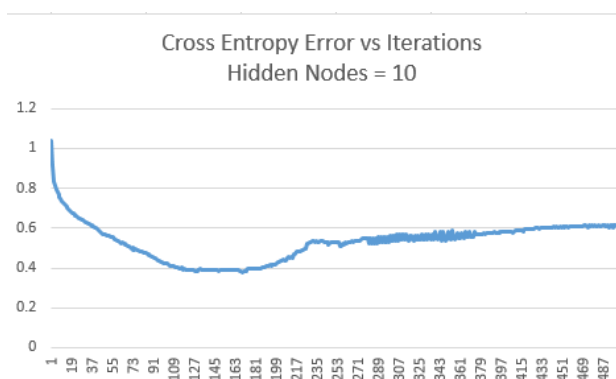
CS559

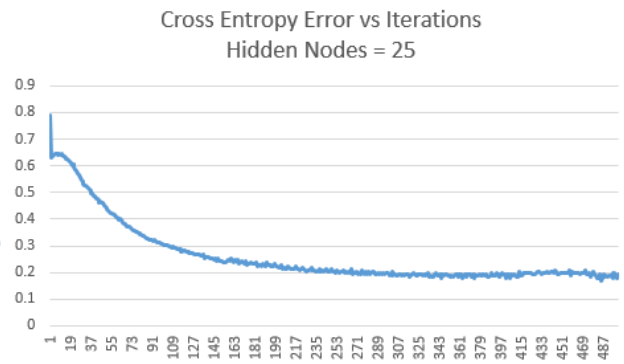
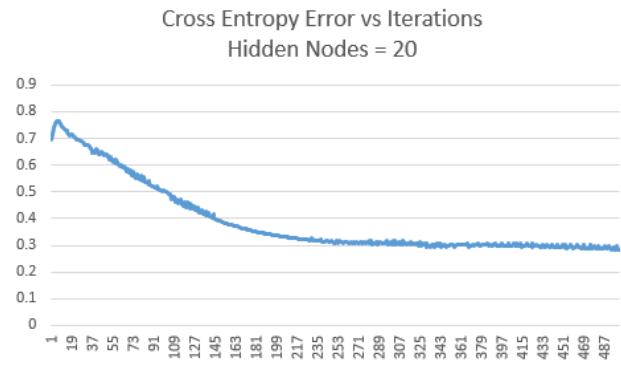
*I pledge my honor that I have abided by the Stevens Honor System.*

## HW2 Data and Observations

Our assignment in homework 2 was to program a neural network with back propagation to help predict two random datasets built from the same rules. Tuples ranging from 0 to x were created, and then an output dataset was created by adding up the floored values of the tuples, and modding by 2. Ie,  $[0.5, 1.2, 1.2]$  became  $0 + 1 + 1 = 2 \% 2 = 0$ . The neural network's goal is to learn how to predict the output from our input, and then apply its found weights onto a test set in the same form. Different quantities of hidden nodes, different learning rates, and different quantities of epochs yielded different levels of accuracy over time.

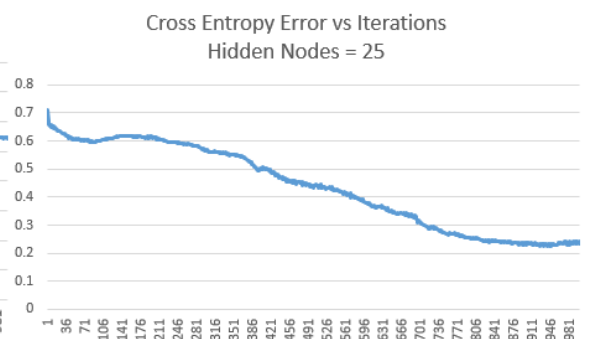
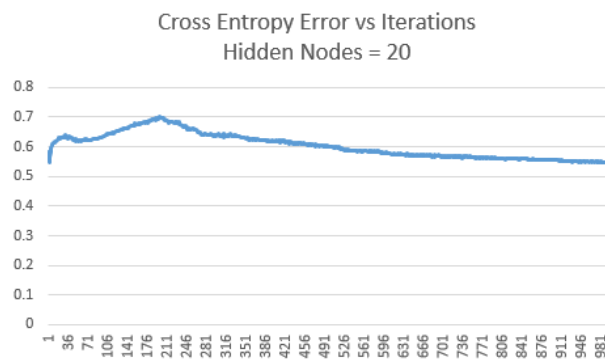
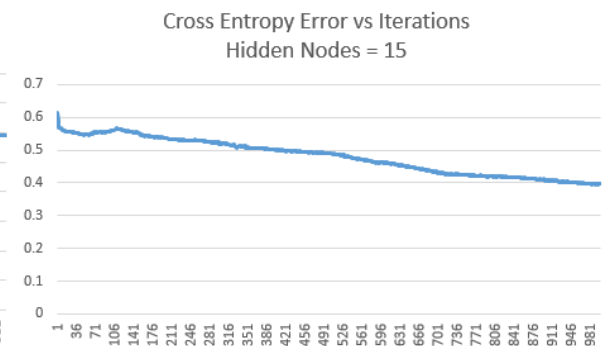
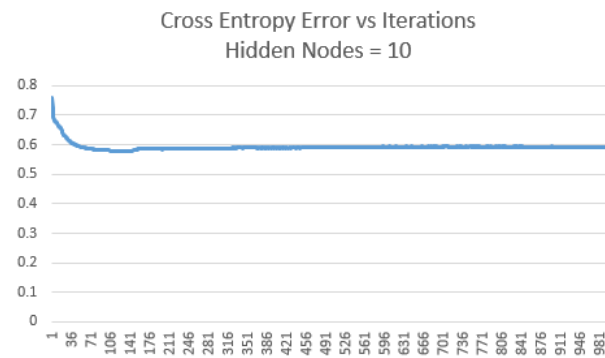
For  $n = 200$  on the dataset from 0 to 2, we can observe cross entropy error descend over 500 iterations with learning rate 0.01. Here is how it starts to converge over different quantities of hidden nodes.





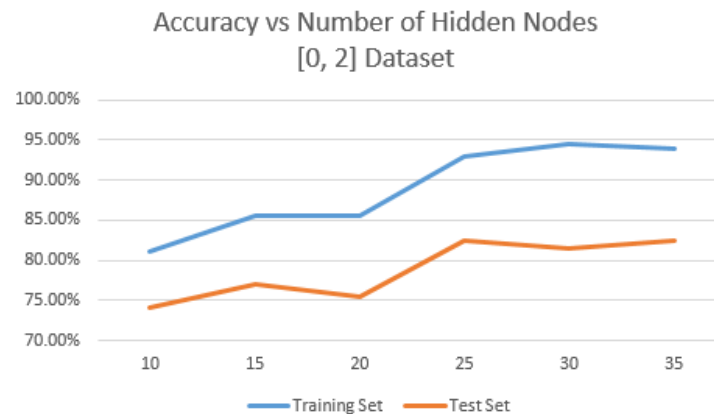
As you can see, they all follow a similar, descending trend as they level out.

Here are similar measurements, using 600 samples on a dataset from 0 to 3, over 1000 iterations.



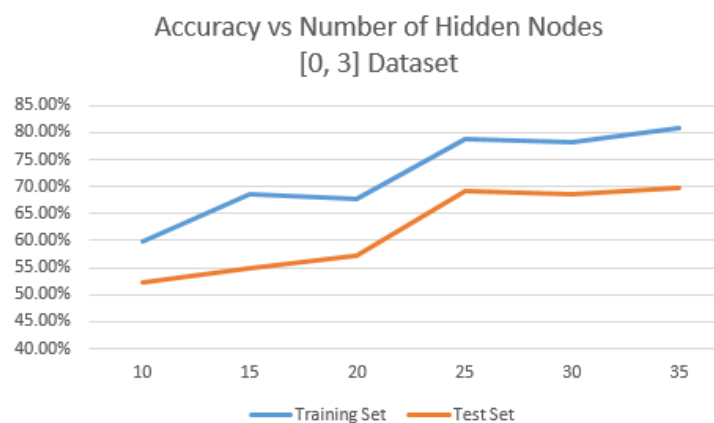
As before, the graphs level out over the 1000 iterations. If we continued over more iterations, we would see them start to reach a horizontal asymptote.

Using the Neural Network to predict over randomly generated datasets yielded different results, depending on the parameters. This graph shows the relative accuracy shown on a training and test set (both of length 200), using a learning rate of 0.01 over 500 iterations on the  $[0, 2]$  dataset.



As we can see, adding hidden nodes helps up to a certain point. We start seeing minimal gains after around 25 nodes.

Similarly, here is a graph of the accuracy shown in predicting a training and test set of length 600 using a learning rate of 0.01 over 1000 iterations on the  $[0, 3]$  dataset.



As above, we see adding hidden nodes helps us make more accurate predictions, though we start to see smaller jumps as we add more and more nodes.

## Conclusions

As each input in the beginning dataset is equally important to the output, it didn't surprise me to see generally similar weights throughout the resulting network, changing in each iteration in response to new random datasets and in response to how many hidden nodes they went through. As more hidden nodes added, the network becomes better at weighting itself to a training set, and we see more accurate predictions over both that training set and the test set.

In general, it seems that the network is better at predicting over the  $[0, 2]$  dataset than it is over the  $[0, 3]$  dataset. I am not positive as to the reason for this, though with the  $[0, 2]$  set each input ends up becoming either a 0 or 1 when used in processing the output, which may make classification easier. Different learning rates and epochs also change these numbers, though I kept them consistent for the sake of demonstrating the effect changing a single variable (ie, quantity of hidden nodes) had on the results of the program. It is possible that, with more testing on different quantities of iterations, learning rates, and hidden nodes that we could see 95% accuracy on either dataset. Intuitively, I tend to see very similar results in predicting over a corresponding training and test set, even when the test set is much larger than the training set. An example test on a  $[0, 2]$  dataset with 30 hidden nodes, learning rate 0.01, 1000 iterations, on a training set of 2000 and a test set of 5000 yields 1837/2000 (91.85%) on the training set and 4566/5000 (91.32%) on the test set, supporting this.

## Operation

This program, hw2.py, is a Python 3 script that uses the numpy library. If you do not have it installed, you will need to use PIP to install numpy. After that, in any command window navigated to the right folder, running "python hw2.py" will execute the program.

During runtime, the program will first ask you what dataset you are looking to run this on, from 0 to x. Examples include entering 2, which will generate random values in the  $[0, 2]$  range in the tuples, 3, which will generate from  $[0, 3]$ , etc. The program will then ask how large the training and test sets should be, and will generate accordingly. Next, you will be prompted to enter how many hidden nodes you'd like to use, followed by a learning rate, and a number of iterations to train over. After that, the program will run, predicting over both the training and test sets, will show you the number of guesses it got right, and will output the network to weights.txt. It also outputs the sum\_error over time to outputs.txt, which was used to graph cross entropy earlier in this report.