Edward Venator

EECS 484 Assignment 3: Maxnet

Basic Methodology

I began by filling in the appropriate weight matrix and excitation function equations. The weight matrix is a matrix filled with the inhibition strength ε, except for the diagonal, which is filled with ones. The excitation function simply 0 for inputs less than 0 and returns 0 for inputs greater than 0.

Once I had this code working, I modified it to run without human intervention by changing the end condition to stop the code when there is only one non-zero sigma value. If there are no non-zero sigma values, the code notes that the network failed. I then removed all plots, prompts and outputs, creating a routine that could be run quickly and without any user intervention. I then wrote a wrapper script to run the maxnet routine repeatedly with different values of epsilon and number of nodes. For each combination of parameters, the inhibition strength, node count, run time, and success were dumped to an Excel spreadsheet. I later added averaging of ten trials with each set of parameters to cut down on noise from the random initial conditions.

Findings and Conclusions

A Lower Bound for Inhibition Strength

My first finding was that there is a very concrete definition for the largest inhibition value that will guarantee a solution. When ε is bounded by this inequality, the network is guaranteed to return the maximum node for any input.

$$\varepsilon \geq -1(n_{nodes} - 1)$$

After I discovered this inequality, I added it to my wrapper to restrict my results to only those guaranteed to be successful. If some information was known about the inputs, this inequality could perhaps be altered to allow for larger values of ε (and therefore faster results), but this inequality guarantees an output for any input.

Optimal Values of Inhibition Strength

I then proceeded to run the wrapper script for n_{nodes}=2 .. 100 with ε stepped by increments of .001 through the entire allowable range, as defined by the inequality above. Making plots of the output, I observed the relationship between the inhibition strength and the run time.

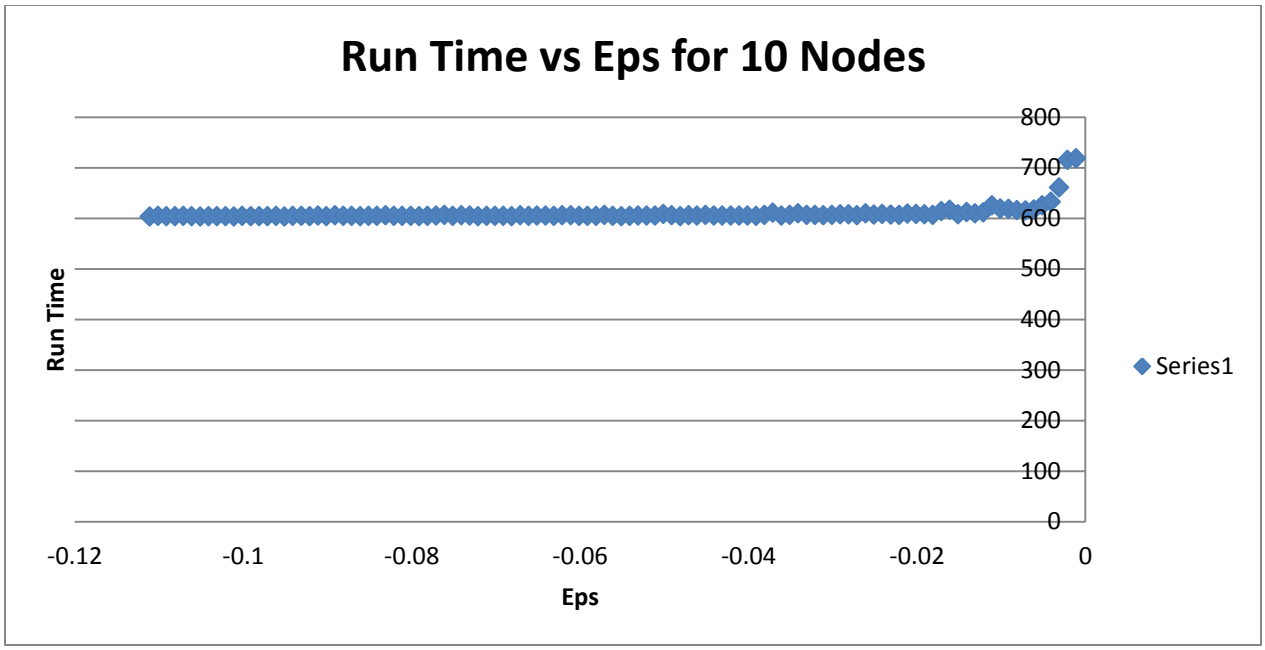

Figure 1: Run time for various inhibition strengths on networks with 10 input nodes.

In figure 1 above, it is clear that decreasing the magnitude of ε is not very significant until $\varepsilon > -\frac{1}{5*n_{nodes}}$. I also observed a similar inflection point for 20 nodes and 50 nodes. From this data, I conclude that a good heuristic for setting the inhibition strength is that in can be anywhere in the range defined by this inequality without any risk of the algorithm failing or noticeable difference in runtime for evenly distributed random inputs.

$$-\frac{1}{n_{nodes}-1} < \varepsilon < -\frac{1}{5 * n_{nodes}}$$

The Effect of the Number of Nodes on Run Time

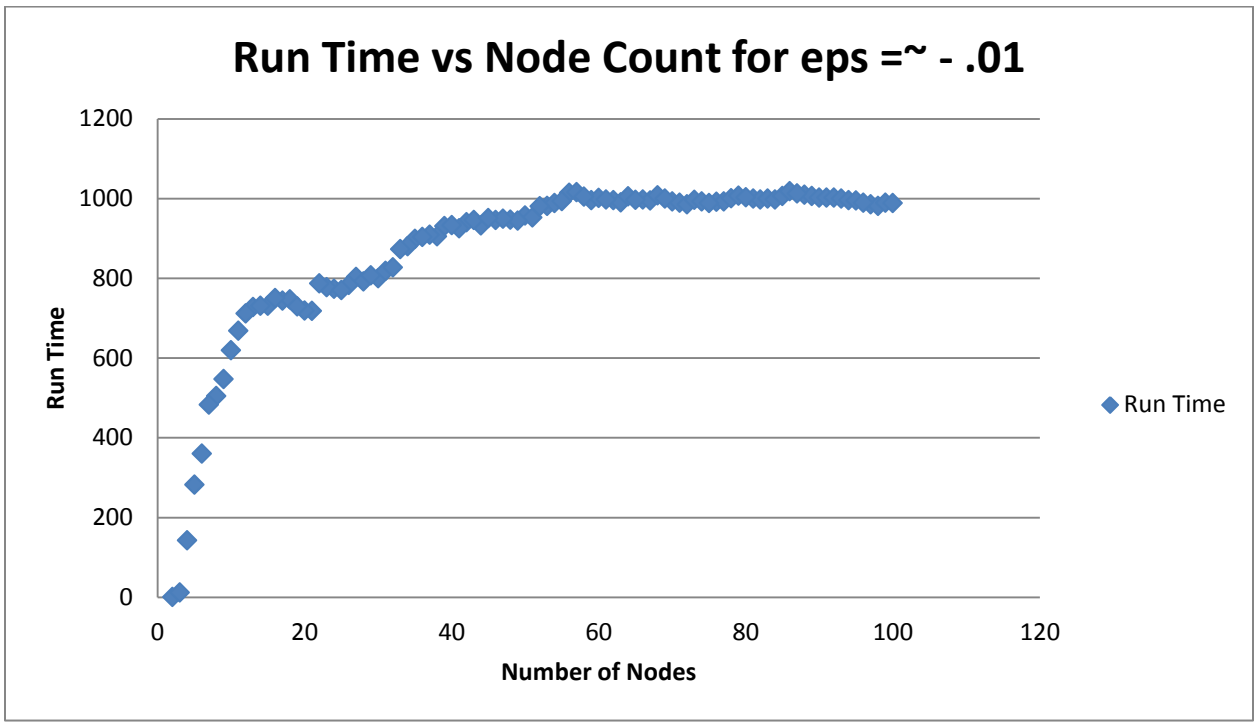

Figure 2: The effect of the number of input nodes on run time

When the inhibition strength is held constant, it is very easy to see that the number of nodes is a strong factor in the run time of the algorithm. Figure 2 above shows this relationship for an inhibition strength of about -.01. Interestingly, the run time begins to level off at around 60 nodes. I suspect that this is because of the inequality described above. .01 is in the range defined by that heuristic for any number of nodes between 20 and 100, which is consistent with the results shown in figure 2. Although figure 2 does show the advantage of decreasing ε further, to about $-1/(2*n_{nodes})$.

The Effect of Different Types of Input

I did not extensively explore different distributions of input. Mathematically, the scale of the input is irrelevant; the algorithm will run in the same time as if the input data were scaled to the 0 .. 1 range. However, the distribution of the data would affect the run time. Input points that are very close together relative to their magnitude would take much longer to run to completion. If there is more than one input with the same maximum value, my current end condition (one node remaining) would not be fulfilled, and the code would keep running until there were no nodes remaining, resulting in the code reporting a failure. This never happened in my tests, but the odds of this happening with random data are very slim, whereas it could be very likely with some types of inputs. For these inputs, a different end condition would be necessary, such as ending when all node outputs are equal.

Appendix A: Attached Code and Data

- activation_fnc.m Is the activation function, which is a linear function, saturated at 0 for inputs less than 0.

- maxnet.m is the maxnet main program, based closely on Professor Newman's original code. It is designed for human use. It has been modified to initialize eps, W, and node_count, to change the names of some variables, to run faster, and to run to completion without human intervention.
- auto_maxnet.m is the maxnet routine, based on maxnet.m, but with all inputs and outputs removed to make it faster and so that it can be run in a wrapper script.
- automate_maxnet.m is a wrapper script to run auto_maxnet.m repeatedly and dump the results to an Excel spreadsheet.
- results.xls are the results of running automate_maxnet for ten runs each, averaged, of all guaranteed successful eps values in .001 increments for all possible numbers of nodes from 2 to 100.