James Boulter

Intro to CogSci Final Project Write-up

Professor Bram Van Heuveln

Neural Networking Visual Model Simulator

After many hours of programming and debugging, I have created a working neural network simulator inside of C++/Qt. I chose to write in C++ because it is the language I am most proficient in, and I chose Qt because I needed to learn to use it, and it pairs well with C++. It also allows deployment on many platforms. Since I use both OS X and Windows, this was important to me. Overall, I am pretty happy with the fruits of my labor, given the hours I put into it, which I would estimate to be around the same as my average Data Structures homework at 30 to 40 hours. I chose to implement a neural network because I thought it would be the most fun option for a final, and I believed it would not take very long. I was right about it being fun and interesting, but it took much longer than expected.

My goals for this software were clear, from the start. Basic functionality includes taking in a file containing data for a neural network, processing that data and assigning it both visual and non-visual properties, and the ability to query the network. I accomplished all of my goals with this project and I plan to come back and add extra features in time, which will include the ability to query with multiple terms (easy), the ability to adjust the number of outputs and activation/deactivation thresholds (easy), adding a reset button (moderate), and the ability to see which nodes are being processed as they process (hard). However, I believe the software is still currently very much functional.

To query the network, type in the box on the bottom and hit Query. This will trigger the chain reaction we expect in a neural network, activating some nodes and deactivating others. As nodes become more active or deactive, they influence their connections more. So a node that has been activated a lot has a lot more influence over its connections than one that has not seen activation. Deactive nodes will deactivate their positive connections, causing things clearly not associated with the query to reach a deactivation threshold, while the positively weighted nodes will activate their positive connections until they reach the set activation threshold. This triggers the object to turn red, and if the object is an entity, its associated name will turn yellow, for easy identification.

On the technical side of things, I wrote 500 lines of code, double the usual Data Structures homework, spread over two classes and a main function. On startup, the main function processes the input file, passed in as an argument to the executable. Through a GUI class, a window containing a graphical view is created. This graphical view is then loaded with spaced out clusters of objects. Objects in the same cluster are negatively connected, although there is no visual connection, to reduce clutter. Objects connected with red lines are positively connected. The green objects are property nodes, meaning they express an attribute that can be associated with an entity. Entities, which express a *thing* with which properties are associated, are blue.

The classes I used to implement the network simulator are the Node and mainWindow classes. Node is the object represented by the visual rectangles. It represents a property or entity and holds all information related to its type of object. The mainWindow is the class holding code for creating and operating the window. This holds the Query method and all visual tasks.

The results of my experience programming a neural network are that it is much more difficult than I originally expected (mostly the visualization part). Displaying the network in a manner that allows it to be human recognizable is difficult, and required a nice algorithm. Networks are very intricate and creating software which can actually parse a file and create a network from it is challenging. Overall, though, the network simulator is functional and demonstrated the things we explored in class. These things being that I can type in a property and see things associated with that property. When disabling entity correlation, that is making it so only properties are shown as output, the network shows the sharkiest sharks and the jettiest jets. That is all I wanted my network to do, and it looks beautiful doing it.

Looking back, if I had known I would be writing 500 lines for this assignment, I would probably have picked something else. However, now that I am finished, I am proud of my work and I am happy with my results.