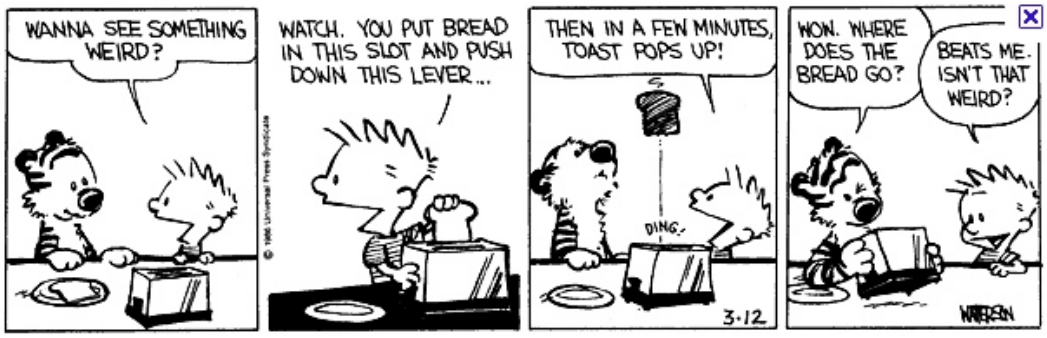
CS311 - Neural Nets Lab
Thursday, April 11



In class today we're going to be playing with a software that simulates neural networks to get a better feeling for how they work, how they can be used to solve problems and how they might mimic the human neurological system.

1 Getting Started

The software that we're going to use is called Simbrain (`http://www.simbrain.net/`) and is available for download at:

`https://code.google.com/p/simbrain/downloads/list`

Download this software and get it up and running:

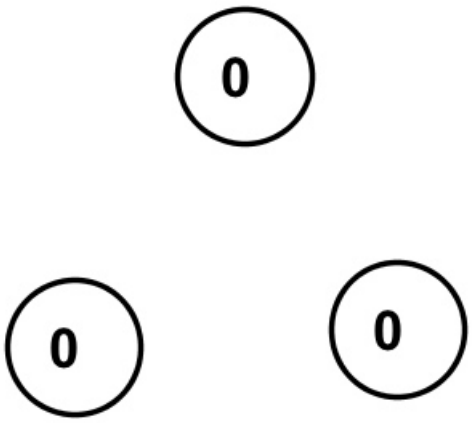
- Go to the url and download the Simbrain3_beta_1.zip somewhere.
- Unzip the file
- Open up a shell/terminal, then traverse to the unzipped directory and run the program:

```
java -jar Simbrain.jar
```

2 The Basics

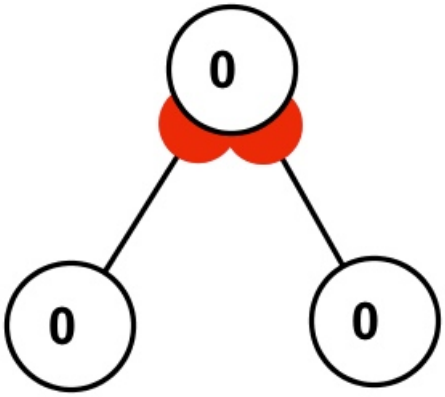
To start with, let's build some simple neural nets. To build a new network, you can either select `Insert` → `New Network` or click on the "new network" button (it looks like a network of nodes). This will bring up the network creation screen.

To add new nodes, click on the "add node" button (it looks like a circle with a bit plus on it). Click it three times to add three new nodes. You can drag nodes around into any configuration that you'd like. To start with, we're going to try and do an "OR" neural net, so drag them into a configuration that looks like:



Now, let's connect up the nodes. To make a connection you first have to specify one (or more) "source" nodes, i.e. where the signal will flow *from*. To specify a node as a source node, click on it (it will be circled in a green box) then press the '1' key. This will circle it in a red box indicating that it is a source node. To create a link, click on the other node you'd like to connect it to then right-click and select `Connect Neurons` → `One-to-One`. In the menu that pops up, just click ok.

After doing this you should see a connection from the source neuron to the target neuron. You can tell the direction of the connection because the target node will have the synapse on it (the red blob). The color of the synapse indicates whether the connection is positive (stimulates) indicated by red or negative (inhibits) indicated by blue. Add two connections so that your network looks like:



We have the structure of our network, now let's define the weights and thresholds. We're going to try and make an "OR" node, so where the edge weights are 1 and the output neuron uses a

threshold of 1. To update the edge weights, double-click on a synapse (i.e. the big red ball at the end of a connection). This will open the "Synapse Dialog" window. For now, the only thing we need to worry about is the "Strength" parameter, which is the weight of that connection. Change the weight to -1 (indicating an inhibitory relationship) and click "OK". Notice that the color of the synapse changes from red to blue.

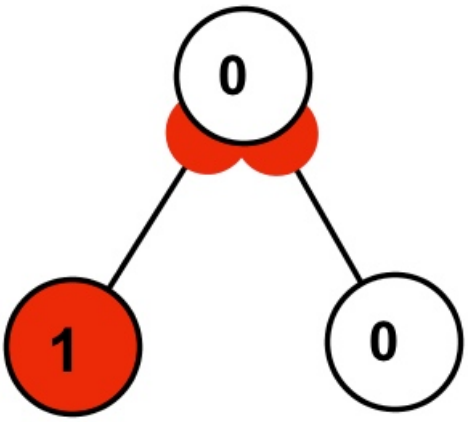
For our "OR" network, we actually want both weights to be 1, so go back in and change the strength/weight back to 1.0.

The last thing we need to check is the threshold on the output node. Double-click on the neuron at the top to bring open the "Neuron Dialogue". Here is a quick breakdown of the important fields:

- `Activation`: The amount of signal/activation currently at this node (right now, 0.0).
- `Upper/Lower bound`: Maximum and minimum activation allowable from this node.
- `Update Rule`: How the neuron goes from its input values to its output value, i.e. its thresholding function.

We'd like to use the step threshold (all or nothing) which is labeled "Spiking threshold". Select that from the drop-down menu and then set the threshold to 0.9 (this version of the threshold is $>$ not $\geq$) and click "OK".

Now our "OR" network is all set, so let's test it out. The bottom nodes represent the input to our "OR" node, so to test it out we need to put some activation on those nodes. You can do this by either double-clicking on a node or by click on a node and then using the up and down arrows to adjust its activation. Adjust the activation of one of the nodes to be 1.0 so that it looks like:



Notice that as you add activation to a node it changes color accordingly.

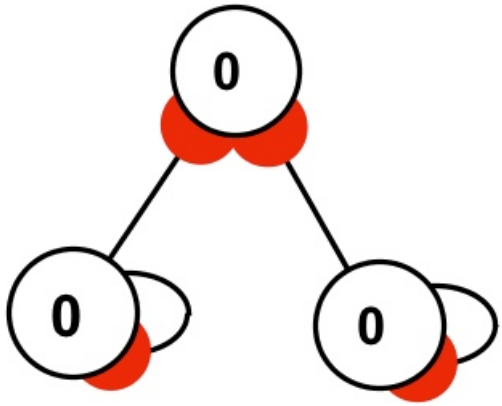
Now we can simulate what will happen in the network. When we simulate a network the simulator will run in steps. At each every neuron that has activation will send its activation along the synapses to any connected neurons. There are two ways to run the network. We're going to just run the

network a step at a time. To do this, just press the "pause" button once. This will step the network forward one step. You'll see the bottom neuron "fire" passing its activation along the synapse to the top neuron. The top neuron will then check its threshold function and then activate since the threshold is met. If you press the step button again, you'll see the activation disappear.

Try out a few different combination of inputs to make sure that our network actually implements the "OR" function properly.

Another way of running the network is with the play butting which will run the network step after step. If you do this when the network has activation you'll see the top node flash red for just a moment (i.e. step) and then go back to 0 because the activation will have been passed along in the previous step and will not be there.

It can be annoying to have to keep refilling the "inputs" to check different outputs. Another way to setup the network is to connect up the input nodes to themselves. This way, whenever they fire, they will pass on activation to other neurons, but they also will keep whatever activation they have. You can do this by selecting a neuron as both the source and the target and then adding a connections, something like:



Now, notice that if you add activation to one of the input nodes and then step multiple times it stays there. When you're network is setup like this, you can then hit the play/run button and alter the input neuron activations on the fly (again, using the up and down arrows) to see the output dynamically. *Again, try out the values and see that it does in fact implement the "OR" function assuming that the inputs are binary.*

Once you're comfortable with all this, try and build a few other networks and test them out:

- *AND: This should only be a small change to the current network.*
- *NOT: You'll probably need to bring up a new network.*
- *XOR: This will involve another layer in the network and more synapses.*

Note that you can save any of your networks to a file if you'd like to be read back later by the program.

3 NNs and the Brain

To understand and visualize a bit how the brain (or a neural network) can perform parallel computation, we can play with a massively parallel network and watch what happens as activation propagates throughout the network.

The Simbrain package comes with a few pre-loaded networks that you can open. Go to `File` → `Open Network` to open a save network then open the file in `simulations/networks/IntFireProbSynapsesAvalanches.x`

Look at this network (including opening up some of the neurons) and try and figure out a bit of what's going on with the network. You may need to look at the documentation online, though don't spend too much time on this.

Now, let's see what happens when we activate some of the neurons and then propagate the signal:

- Select the bottom neurons.
- Click on the randomize button (it looks like a die). This randomized the activations of the neurons.
- Click somewhere else on the window to unselect the bottom neurons.
- Step through and watch the activation propagate through the network.

Repeat this a few times. If you'd like, feel free to press the run button and watch the propagation happen in "real time".

What are your observations?

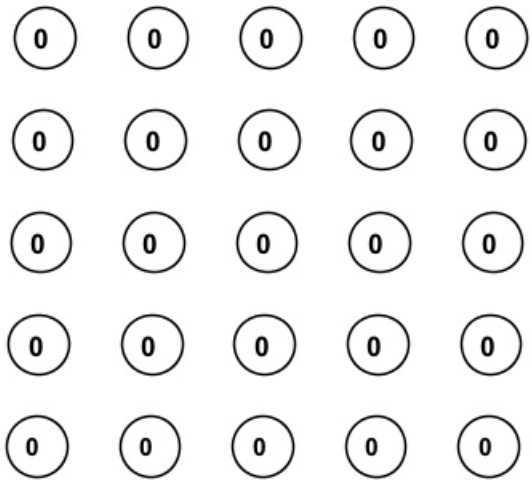
4 Identifying Digits with a NN

One of the early successes of neural networks was their use in identifying handwritten digits. Given an image of a digit where all the extraneous information has been cropped, the basic setup is as follows:

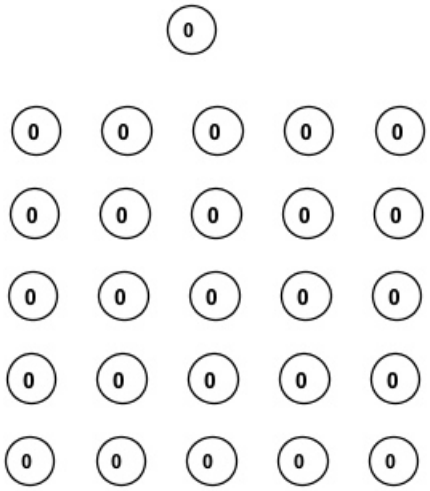
- Break the image into a grid.
- Represent each square in the grid as an input to a neural network.
- The amount of black in the grid square represent the signal (in the extreme, you should just make it a 0 or 1 depending on if there was "enough" black in the square).
- The neural network then uses this information to decide what digit it is.

To start with, we're going to do a simple variant to try and identify whether a "digit" is a '1'.

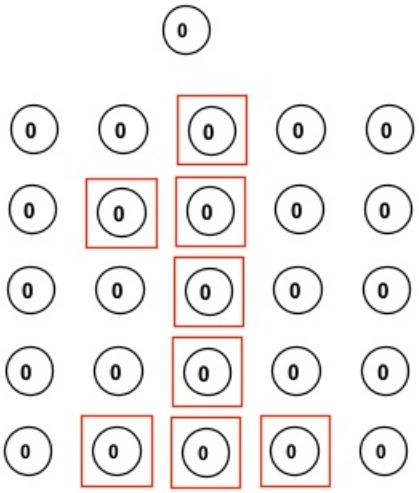
First, setup a grid of neurons, say a 5x5 grid like:



These will represent our input grid. For now, let just use a simple perceptron (i.e. a single neuron) to decide if the input grid is a 0 or a 1. Add one extra neuron and move it to the top. This will be our deciding neuron:

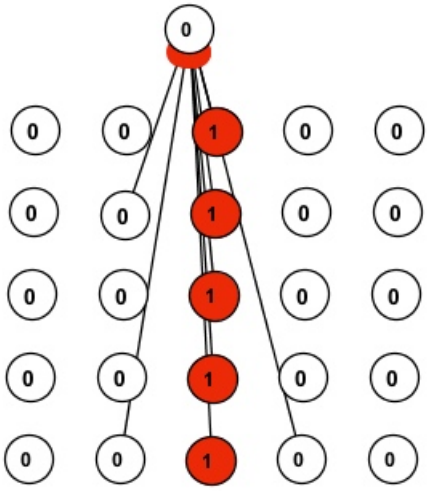


Now, we'd like to connect up all of the neurons that would result from a '1' being shown. Holding down the shift key select the neurons below and then hit 1 to make them source nodes:



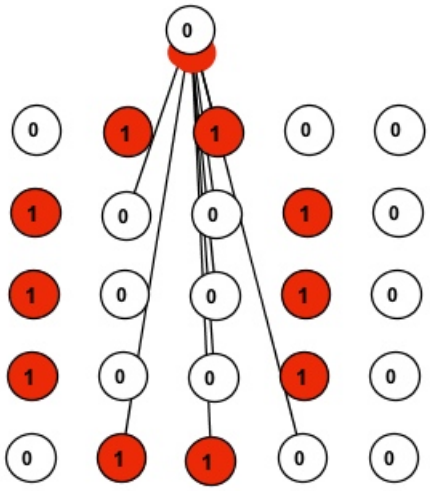
We'd like to connect all of these neurons to our top neuron. Select the top neuron and then right-click on it and select `Connect Neurons` → `All to All` and then click "OK". This will connect *all* of the source neurons to *all* of the target neurons, though in this case there is just one target neuron. Now, we'd like to set the neuron threshold appropriately to classify the digits. One option is to set all the synapse weights to 1 (i.e. leave them as is) and then set the neuron to be something like 4.9. Feel free to try out different values if you'd like. You can select all of the synapses and set them at once by dragging encircling all the synapse ends (red blobs) and then double-clicking on one of them.

Now that we have this, we should be able to classify a new digit "input". For example you can enter a basic "1":



and then press the step button and the upper neuron will classify at a 1 (if you hold down shift and then select multiple neurons you can use the arrow keys to increase/decrease the activations of all of these nodes at once).

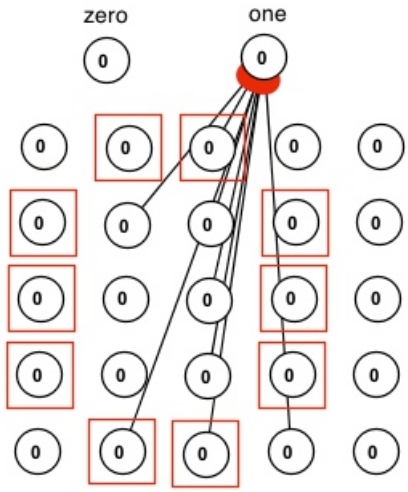
Now if you enter a "0":



and press the step button the network will not activate, indicating that it is NOT a '1.

Try out a few other numbers on this network. Do any of them falsely fire? Are there any variants of a '1' that don't register? Tweak the weights to minimize your misclassification.

Identifying a single digit is interesting, but what we're really like to do is recognize all 10 digits. We can do this by adding another neuron for each digit. For example, if we wanted to try and identify zeros as well we could add a new neuron and connect to these nodes:



Add the appropriate connections to the "zero" neuron and then try out your experiments again with different digits and see how they do.

If you'd like, try and add a few more digit recognizing neurons and then test out how well your digit identifier works on the different patterns.

5 Chasing Amy

Simbrain also allows you to "deploy" your networks in "real-world" scenarios to make decisions. Doing so is more than we'll have time to do in one class, however, Simbrain does come with some pre-made simulations that you can play get a feeling for how you could deploy your networks.

Go to `File` → `Open Workspace File` and then open the file `simulations/workspaces/chasingAmy.zip`.

A workspace is a collection of windows within the Simbrain framework including both networks as well as "worlds".

Take a look at the vehicle.xml network. What does it do? How does it work? The top nodes are output nodes and the bottom nodes are input nodes.

Once you have a feeling for what it does, run the simulation. When the simulation is running, you can drag around the people and see how the simulation reacts.

Play around with the simulator some and see if your intuitions about how the network behaves were right. To get more detailed input/output stop the simulator, move the people around and then step through a couple of steps.

6 More fun things to try

If you finish all of this, use your remaining time to play with the simulator some more. You may do whatever you'd like (as long as it's constructive :), but here are some things you could investigate:

- Try and code up some more interesting networks, for example try and a network that "remembers" or see if you can encode a tic-tac-toe player.
- Play a bit with multi-level networks and see how their behavior works.
- Open up some of the other saved workspaces and see if you can figure out what they do.
- The online documentation is spotty, but see if you can create networks that interact with the "real-world".