

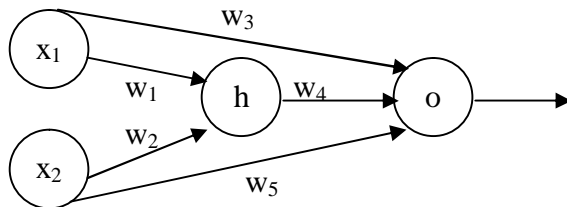
## Convergent Learning in Neural Networks

Assignment due: December 10<sup>th</sup> at 11:59:59 p.m.

There are some neural network topologies (connections between neurons) that fail to produce correct output for a given problem no matter what settings are made to the weights. Similarly, even for topologies for which solutions do exist, the back-propagation weight-adjustment algorithm is not guaranteed to arrive at a solution.

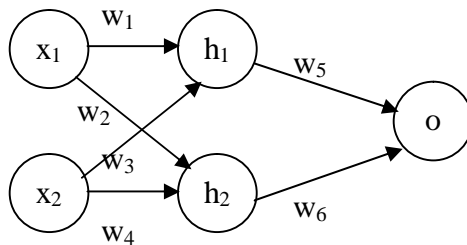
Your task is to examine three topologies and run a series of experiments on them to determine which topology is best, and by how much. All three topologies are attempting to learn the XOR function and all weights are to be initialized to a random value uniformly chosen from the range  $[-1,1]$ . Training and testing will use different functions – training will be done with the sigmoid function ( $\text{sigmoid}(x) = 1 / (1 + e^{-x})$ ) and testing will be done with a simple step function whose threshold is 0 ( $\text{step}(x) = (x > 0) ? 1 : 0$ ), where  $x$  represents the net input to the hidden or output neuron. The net input is the sum of the product of each input times its corresponding weight. For boolean inputs, 1 should be used for true, and 0 for false.

Topology 1 is the simplest topology you will be using:



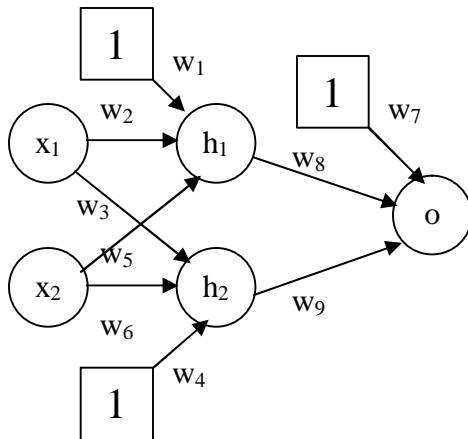
All thresholds are 0 for this topology and do not change.  $w_1$  through  $w_5$  are initially chosen randomly from the range  $[-1,1]$  but do not necessarily need to remain within this range.

Topology 2 is as follows:



All thresholds are 0 for this topology and do not change.  $w_1$  through  $w_6$  are initially chosen randomly from the range  $[-1,1]$  but do not necessarily need to remain within this range.

Finally, topology 3 is:



The weights coming from the inputs labeled “1” act as thresholds that can be learned from back propagation. Note, however, that these weights are actually the negative value of the corresponding threshold. E.g., for hidden neuron  $h_1$ , the neuron fired during testing if  $w_1 + x_1w_2 + x_2w_5 > 0$ , or  $x_1w_2 + x_2w_5 > -w_1$ . Of these two equations, the former equation is most likely the one you will want to use for your code.

For each topology you should run 1000 tests (with different initializations for the weights) to estimate the probability that a randomly chosen initialization will be able to learn the correct weights. For each training session you should let the network iterate for at least 10,000 iterations before you declare it incapable of learning. Once it has learned acceptable weights, however, it should stop iterating for that test.

Remember that not all topologies have acceptable weights! Before running your simulation you might want to see if you can manufacture weights that give the correct answer.

The output for your code should be as follows:

```

** Topology 1 **
Number successfully taught:
Number of attempts:
Estimated probability of random success:
Maximum number of iterations required for successful test:
** Topology 2 **
Number successfully taught:
Number of attempts:
Estimated probability of random success:
Maximum number of iterations required for successful test:
** Topology 3 **
Number successfully taught:
Number of attempts:
Estimated probability of random success:
Maximum number of iterations required for successful test:

```

The deliverables are similar to previous programming assignments. All code necessary for execution should be included, as well as a makefile if appropriate. Additionally, a

readme.txt (or readme.doc) file should provide a discussion of your results, any thoughts you have on the assignment, and any special requirements for execution of your code (there should be none, but if there is – let us know). All of these files should be zipped up into either NNxor.zip (Windows) or NNxor.tgz (Linux/Unix). The executable should be named either NNxor.exe (Windows) or NNxor (Linux/Unix).

Please get started early and ask questions as they arise. Don't hesitate to ask clarification questions. The TA (Ben Hocking) has completed a solution to verify the time/complexity of the assignment isn't too great.