

Artificial Neural Networks: A Programmer's Notebook

By Thomas J. Clancy

Monday, December 01, 2003

11:52 PM

I've spent several weeks, on and off, learning about artificial neural networks and, thus like my GO-writing days, I am attempting to put down on paper my understanding of artificial neural networks (ANN from now on).

I have been reading, albeit slowly, a book called "Fundamentals of Neural Networks: Architectures, Algorithms and Applications," by Laurene Fausett. There is no code in the books, but rather it is filled with diagrams, explanation of the kinds of ANNs out there and the algorithms that they employ. It is also heavy in mathematics, but I have been slowly pressing my way through, trying to understand.

As I've been going along in the book, I've been thinking about how to go about writing a general library of classes that would allow me to model the kinds of ANNs and then execute them given some input data. So far I've designed a crude little class library that contains the following classes: Edge, Neuron, Activation, and Network. The Network class is not completed or working yet, but I have been able to, by hand, string together Neurons and Edges with simple Activation objects and get something to work.

A little bit of theory

The following diagram shows the general architecture of a McCulloch-Pitts Neural Network. The weights on X_1 and X_2 are said to be "excitatory," while the weight, -1 , coming from X_3 , because it is negative, is said to be "inhibitory."

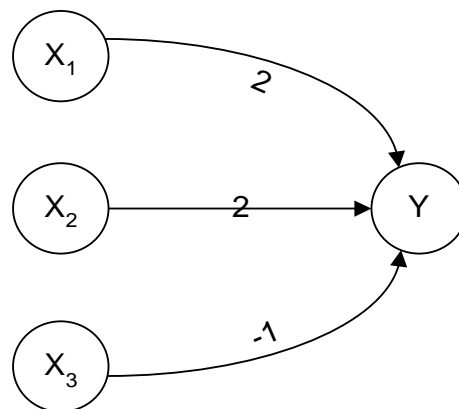


Figure 1. McCulloch-Pitts General ANN Architecture.

So far I can model McCulloch-Pitts neurons for solving simple binary logic functions (AND, OR, AND NOT, and XOR). Here is a diagram of an XOR network. Its interesting to look at.

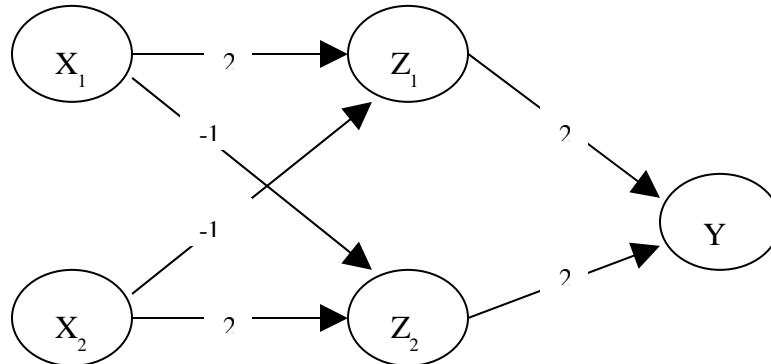


Figure 2. A McCulloch-Pitts neural network for XOR function.

Z_1 , Z_2 , and Y each have a threshold of 2. If you follow, in turn, the input bits for the XOR truth table:

a	b	→	C
1	1		0
1	0		1
0	1		1
0	0		0

Thus ‘a XOR b’ where a=1 and b=1 yields 0. If you input a into X_1 and b into X_2 and run the network, Y will yield 0 when completed.

The routines I’ve am designing work with the different layers. For example, X_1 and X_2 are layer 0, or the input layer (I’ve read many documents online and such that do not treat the inputs as neurons and thus layer 0, in their models, would begin at Z_1 and Z_2 . In the book, Fausett talks about the inputs as neurons, and so I’ve taken to viewing them as a layer. Thus while the diagram above shows a 2-layer network, my layer architecture would have three layers.

Originally I had a separate vector of input neurons, but then the code began to look a little more complicated than it really was. The Activation class is where some of the magic occurs. Essentially the Activation function for the McCulloch-Pitts neuron states the following: $f(x) = \{1 \text{ if } x \geq \theta, 0 \text{ if } x < \theta\}$, where θ is the threshold.

Tuesday, December 02, 2003

8:34 PM

This evening I finished the Network object that allows one to build a neural network and run it once through. Of course this is assuming we're working with the McCulloch-Pitts ANNs. The run method fires each neuron in each layer, from layer 0 to layer N. When completed you can check the output of the any of the output neurons, in the case of Mc-P nets, there is usually only one output (at least as far as I've seen in the book).