Artificial Neural Networks Self-Check

605.645 – Artificial Intelligence

The purpose of this self-check is to make sure you understand key concepts for the algorithms presented during the module and to prepare you for the programming assignment. As you work through problems, you should always be thinking "how would I do this in code? What basic data structures would I need? What operations on those basic data structures?"

One of the more interesting developments in mathematics is the realization that you can draw a function as a graph (and I don't mean a "chart" or plotting the function).

In particular, you can represent weights on edges and operations/functions on nodes.

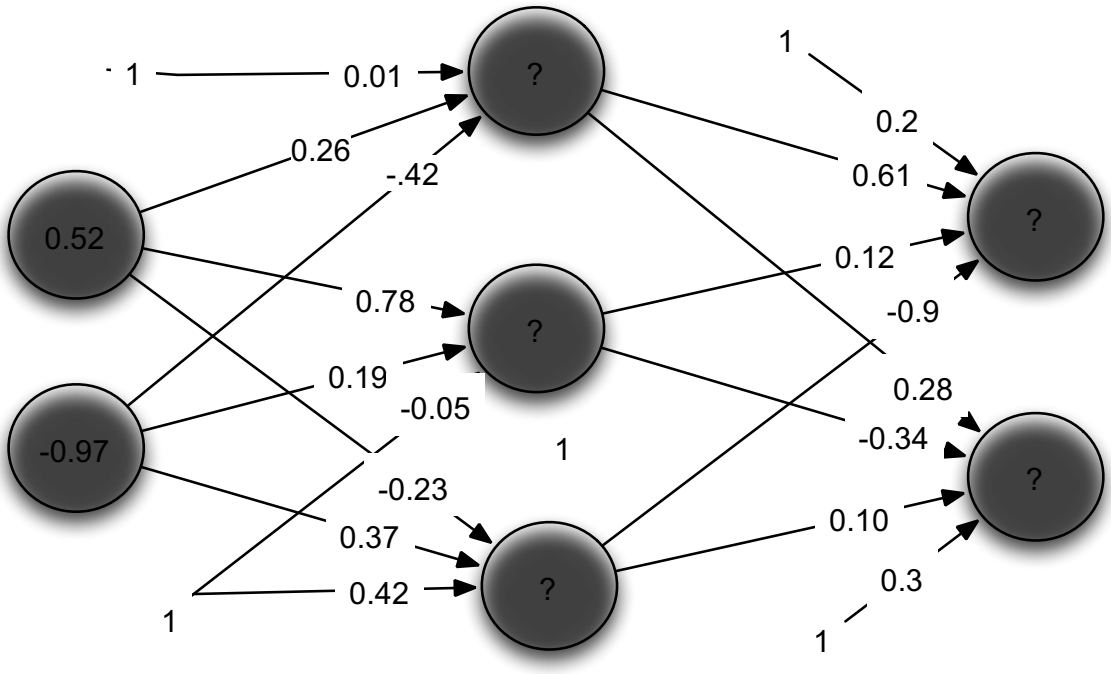
1. Draw a linear regression function of 4 variables (4 x's) and a single output value (y) as a graph. Each x_i will be an input variable. Each edge will have a weight theta_i. You can think of the dot product of the x's and theta's as z and the output function as y = z. Don't forget the "bias" term.

2. Draw the same as above but for a logistic function. What's different?

3. One of the downsides to logistic regression is that we can only make binary classifications. One way of getting around this is to make N logistic regressions, separating the training data into "1" and "not 1", "2 and not 2", etc for each of the N possibilities. You then use these N logistic regressions together and assign y to the value from the logistic regression with the largest value. Assuming 4 X's and N=3, draw this network.

Having gone through the above exercises, neural networks aren't really all that mysterious. Building on the idea of multiple logistic regressions working in parallel, we simply use the first layer of logistic regressions as inputs to the next layer of logistic regressions. The final output can be a linear function (y=z), a logistic function or multiples of these.

4. Given the following neural network with logistic “activation” functions, calculate the output of the network (Calculate all the “?”)



We didn’t really talk about the fact that an observation might have 2 y values. This doesn’t really happen much. You are much more likely to see multiple outputs if you have a classification problem with more than 2 classes.

Assuming that the *true* output should be [1, 0] and that alpha = 0.01, calculate the new weights of the network using the back propagation algorithm. Remember that the backprop algorithm, is essentially the “same” as sending the “signal” back through the network where the signal is the *error*.

Since the goal is to update the *weights*, you can think of the error as going back along the edges. This of the arrows as being reversed on the network and now the error is transmitted back to the weights that created it.

Notice how the weighted error uses weights that were not originally combined in the generation of the output. I've moved the values back towards the originating nodes to emphasize this fact.