Given n inputs,

A = [A1, A2, A3, …, An]

and a single output,

B = k

the importance of a node for the output can be determined by applying the following methodology:

Aabs = k|

for i=0 to len(A)-1:

if B ≥ 0:

Importancei = Ai / Aabs

else:

Importancei = Ai / -Aabs

Where Importancei is the scaled importance of the ith input.

Once the importance has been decided for all for all inputs, this process can be repeated for another output, gathering even more importance for the inputs. After gathering all the importances, they can be added and normalized for each node, to get an expected importance of the node for a particular outcome.

For my application, my input nodes will start at a size of 1. If the node is very against the action being taken, it will approach a size of 0.25. If the node doesn’t really do much for the action being taken, it will approach a size of 1. If the node does a lot for the action being taken, it will approach a size of 4. Size will be determined by the following:

for i=0 to len(Importance)-1:

if Importancei < 0:

size = 1 – (|Importancei|\*3/4)

else:

size = 1 + (Importancei \* 3)

Example:

A = [0.9, -0.9, 0.2, -0.7, -0.3]

B = -0.8

Aabs = 3

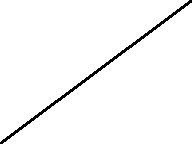
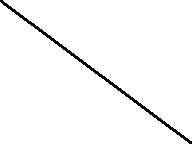
Since B is negative, positive values of A become negative (they are going against the outcome) and negative values of A become positive (they are supporting the outcome).

Importance = [0.9/-3, -0.9/-3, 0.2/-3, -0.7/-3, -0.3/-3]

Importance = [-0.3, 0.3, -0.07, 0.23, 0.1]

Node\_Size = [0.775, 1.9, 0.9475, 1.69, 1.3 ]

Now let’s look at a unique example.



By applying the importance algorithm on the hidden layer, which is connected to the final output, we can get their relative importance for the decision. (Note that all nodes are assumed to have an output of 1, so we only have to worry about the weights in this example). We have an array A = [2, -1], and B = 1. Aabs can be calculated to be 3. Getting the scaled importance, we get Importance\_Hidden = [2/3, -1/3] = [0.67, -0.33]. So, we have the importance values for the two nodes in the hidden layer.

But what about the importance values for the input layer?

First we can figure out importances with respect to the top hidden node. This has a vector of A = [0.5, 4]. Applying the algorithm, we get Importance\_Top = [0.5/4.5, 4/4.5] = [0.11, 0.89]. Doing the same for the bottom hidden node, we get A = [0.3, 0.2] and Importance\_Bottom = [0.3/0.5, 0.2/0.5] = [0.6, 0.4]. However, note that these importances are not yet scaled by the importances in the hidden layer. Importance\_Hidden = [0.67, -0.33], where 0.67 refers to the importance of the top node and -0.33 refers to the importance of the bottom node. Scaling the vectors from before, we get Importance\_Top \* 0.67 = [0.11 \* 0.67, 0.89 \* 0.67] = [0.074, 0.593], and Importance\_Bottom \* -0.33 = [0.6 \* -0.33, 0.4 \* -0.33] = [-0.2, -0.133]. (Note that Importance\_Top[0] and Importance\_Bottom[0] are the importances for Input[0], and Importance\_Top[1] and Importance\_Bottom[1] are the importances for Input[1]). We then add these importances on a node-by-node basis and normalize over the entire layer. Input[0] gets an importance score of 0.074 + (-0.2) = -0.126, and Input[1] gets an importance score of 0.593+(-0.133) = 0.460. Normalizing the layer gives:

Magnitude = |-0.126| + 0.460 = 0.586

Input[0] = -0.126 / 0.586 = -0.215

Input[1] = 0.460 / 0.586 = 0.785

Importance\_Input = [-0.215, 0.785]

Quick example using negatives: Assume the 0.2 and the 0.5 are actually negative. Importance\_Hidden = [0.67, -0.33] as before, but calculating the other importances gets

Importance\_Top = [-0.5, 4] = [-0.11, 0.88] \* 0.67 = [-0.074, 0.593]

Importance\_Bottom = [0.3, -0.2] = [0.6, -0.4] \* -0.33 = [-0.2, 0.133]

Input[0] = -0.274

Input[1] = 0.726

Already normalized since positives and negatives stuck together.

Importance\_Input = [-0.274, 0.726]