Consider some sort of network connectivity model with outputs F_i. Let our total error function look like

$$E=\frac{1}{2}\sum_{I}\omega_i^2 \text{ with } \omega_i=\left(T_i-F_i\right)$$

where

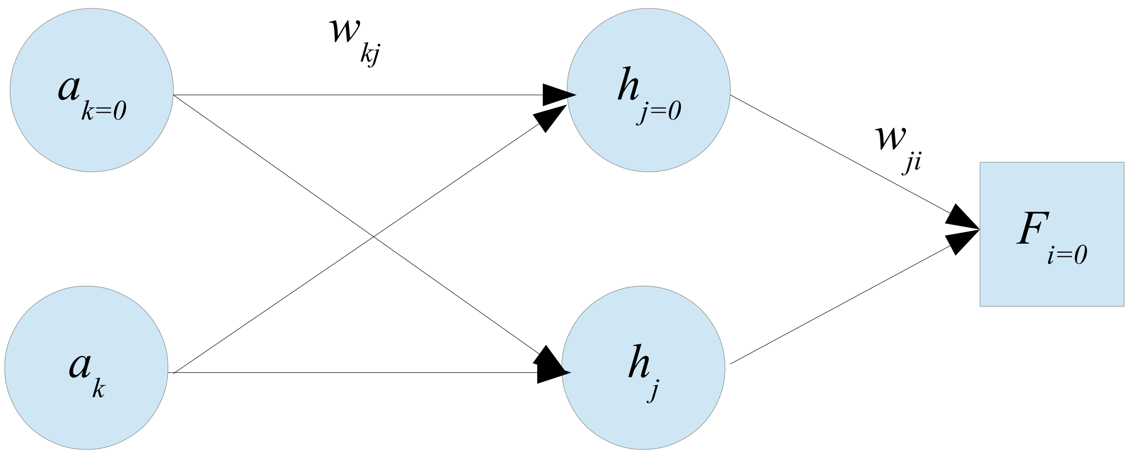
F_I	- Output Activation (what we got): I is the target node index
T_I	- Target Output Activation (what we want): I is the target-node index
E	- Error function for a single training-set model

Where the ½ is there to remove an unneeded factor of two from the derivative and we want to find the minimum with respect to all the weights w (index labels α, β) for all the layers (index label γ), $w_{\alpha\beta\gamma}$, across all the training set data. I am using Greek index variables so as to not confuse this general indexing with the more specific version to be used later. To find a minimum we generally would take the gradient with respect to $w_{\alpha\beta\gamma}$, set it equal to zero and then solve for all the weights through the simultaneous equations. The problem is that there may not be a true zero. In general, we need to find the minimum in E with respect to the weights. We thus have a minimization problem. There are many ways to find a minimum in an N-dimensional phase space. The simplest approach is gradient (steepest) decent, where we go "down hill" in weight space. We modify each weight by the negative derivative and put in a "learning factor", which I am calling λ, to allow us to control how far things can move down hill in each step:

$$\Delta w_{\alpha\beta\gamma}=-\lambda\frac{\partial E}{\partial w_{\alpha\beta\gamma}} .$$

Note that you can also make λ adaptive so that it changes based on how fast things are changing.

Now consider the following simple connectivity model for a single output activation:



For only a single output node:

$$\frac{\partial E}{\partial w_{j0}} = -(T_0 - F_0) f'\left(\sum_J h_J w_{J0}\right) h_j$$

and

$$\frac{\partial E}{\partial w_{kj}} = -a_k f'\left(\sum_K a_K w_{Kj}\right)(T_0 - F_0) f'\left(\sum_J h_J w_{J0}\right) w_{j0}$$

where we will be using the sigmoid threshold function (for now at least) so

$f(x) = \dfrac{1}{1+e^{-x}}$ and so

$f'(x) = f(x)(1 - f(x))$.

Using our theta notation and defining $\psi_0 = \omega_0 f'(\Theta_0)$, where $\omega_0 = (T_0 - F_0)$, $F_0 = f(\Theta_0)$ and $\Theta_0 = \sum_J h_J w_{J0}$ and $\Theta_j = \sum_K a_K w_{Kj}$ we have

$\dfrac{\partial E}{\partial w_{j0}} = -h_j \psi_0$ and $\dfrac{\partial E}{\partial w_{kj}} = -a_k f'(\Theta_j) \psi_0 w_{j0}$.

If we now also define $\Omega_{j_0} = \psi_0 w_{j0}$ and $\Psi_{j_0} = \Omega_{j_0} f'(\Theta_j)$ we get $\dfrac{\partial E}{\partial w_{kj}} = -a_k \Psi_{j_0}$.

We can more clearly see the symmetry if we group things a little differently:

$F_0 = f(\Theta_0)$	$h_j = f(\Theta_j)$
$\Theta_0 = \sum_J h_J w_{J0}$	$\Theta_j = \sum_K a_K w_{Kj}$
$\omega_0 = (T_0 - F_0)$	$\Omega_{j_0} = \psi_0 w_{j0}$
$\psi_0 = \omega_0 f'(\Theta_0)$	$\Psi_{j_0} = \Omega_{j_0} f'(\Theta_j)$
$\dfrac{\partial E}{\partial w_{j0}} = -h_j \psi_0$	$\dfrac{\partial E}{\partial w_{kj}} = -a_k \Psi_{j_0}$
$\Delta w_{j0} = -\lambda \dfrac{\partial E}{\partial w_{j0}}$	$\Delta w_{kj} = -\lambda \dfrac{\partial E}{\partial w_{kj}}$

Before we dive into the XOR problem, let's solve for AND and then for OR using the simple threshold function $f(x) = x$. Figure out the error functions for AND and OR, implement them in your spreadsheet, and then in the programming language of your choice (this latter code will need to be configurable and is what you will eventually be submitting when we generalize it). You will need to specify the initial learning factor, the learning rate change (if you have adaptive learning), the range of random weights, the error threshold that will end the training, and the maximum number of iterations (for timing out).

For the AND and OR networks calculate the six Δw values for each weight in the network and apply them, iteratively, for each training model and observe the convergence. Now try this same process for the XOR connectivity model. This experimentation should be done in your spreadsheet in order to validate the functionality of your Java code. Now change the activation to a sigmoid and play!