Consider some sort of network connectivity model with outputs F_i. Let our error function for one test case, also known as the cost 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 case

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 β, γ) for all the layers (index label α), $w_{\alpha\beta\gamma}$, 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. Ideally we would like the error to be zero. 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. You could apply the Newton–Raphson method to converge on the correct weights. For a single variable function we can find a zero using

$$x^1=x^0-\frac{f(x)}{f'(x)},$$

so if the error function is *E* then we can find the minimum by looking for zeros in the first derivative (which is therefore the *f*(*x*) used in the method) to locate a minimum:

$$w^1_{\alpha\beta\gamma}=w^0_{\alpha\beta\gamma}-\frac{\partial E/\partial w_{\alpha\beta\gamma}}{\partial^2 E/\partial w^2_{\alpha\beta\gamma}}.$$

Or you could use the method of Davidon-Fletcher-Powell

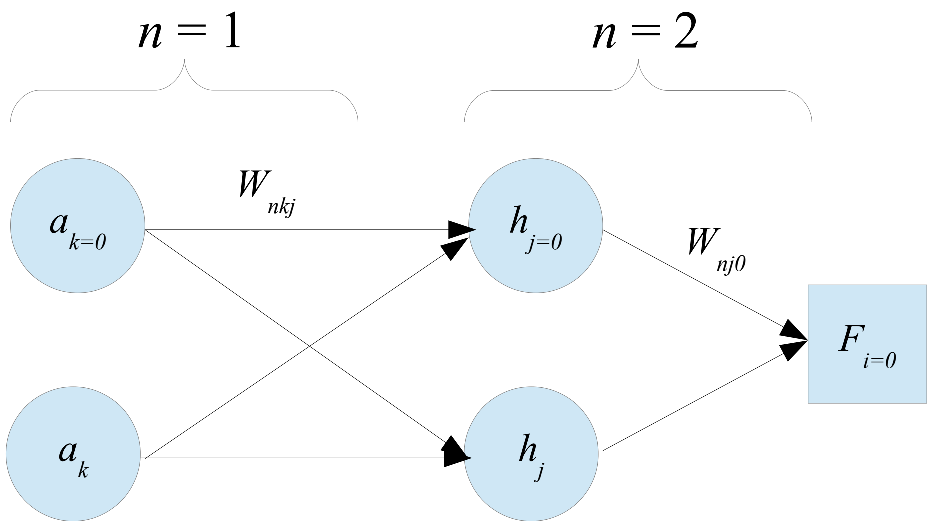
http://en.wikipedia.org/wiki/Davidon-Fletcher-Powell_formula.

The simplest approach, however, 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:



I am going to drop the *n* index for the weights going forward, noting that *n*=1 goes with the *kj* index pair and *n*=2 with *j*0. I am also going to consistently use capital letters for when an index is being used inside of a summation. This convention will make things a bit easier to follow as we move forward. We therefore start with the following relationships:

$E=\frac{1}{2}\omega_0^2$ with $\omega_0=(T_0-F_0)$.

We also have

$F_0=f(\Theta_0)$ where $\Theta_0=\sum_J h_J w_{J0}$

and

$h_j=f(\Theta_j)$ with $\Theta_j=\sum_K a_K w_{Kj}$.

For only a single output node the derivative of the cost function with respect to the weights in the right connectivity layer is straight forward:

$$\frac{\partial E}{\partial w_{j0}}=\omega_0\frac{\partial \omega_0}{\partial w_{j0}}=-\omega_0\frac{\partial F_0}{\partial w_{j0}}=-\omega_0 f'(\Theta_0)\frac{\partial \Theta_0}{\partial w_{j0}}$$

since the T_0 value is a constant. We now need to expand that last derivative:

$$\frac{\partial \Theta_0}{\partial w_{j0}}=\frac{\partial}{\partial w_{j0}}\sum_J h_J w_{J0}.$$

If you look at the above diagram it is clear that the h_j activations are not dependent on the *j*0-weights, so we can write

$$\frac{\partial \Theta_0}{\partial w_{j0}} = \sum_J h_J \frac{\partial w_{J0}}{\partial w_{j0}} = h_j \,.$$

Because that last derivative will be unity when $J = j$ and zero otherwise only a single term in the summation survives. We can therefore write for the right connectivity layer that

$$\frac{\partial E}{\partial w_{j0}} = -\omega_0 f'(\Theta_0) h_j \,.$$

For the dependencies on the weights in the left connectivity layer we begin in a similar manner with

$$\frac{\partial E}{\partial w_{kj}} = \omega_0 \frac{\partial \omega_0}{\partial w_{kj}} = -\omega_0 \frac{\partial F_0}{\partial w_{kj}} = -\omega_0 f'(\Theta_0) \frac{\partial \Theta_0}{\partial w_{kj}}$$

where

$$\frac{\partial \Theta_0}{\partial w_{kj}} = \frac{\partial}{\partial w_{kj}} \sum_J h_J w_{J0} \,.$$

If we look at the diagram yet again we see that the h_j activations are dependent on the *kj*-weights, but the *j*0-weights are not, so we can write

$$\frac{\partial \Theta_0}{\partial w_{kj}} = \sum_J w_{J0} \frac{\partial h_J}{\partial w_{kj}} \,.$$

We can expand that last derivative as

$$\frac{\partial h_J}{\partial w_{kj}} = f'(\Theta_J) \frac{\partial \Theta_J}{\partial w_{kj}} \,,$$

but

$$\frac{\partial \Theta_J}{\partial w_{kj}} = \frac{\partial}{\partial w_{kj}} \sum_K a_K w_{KJ} \,.$$

Again you should recognize that the a_K values are not dependent on the *kj*-weights so we can write

$$\frac{\partial \Theta_J}{\partial w_{kj}} = \sum_K a_K \frac{\partial w_{KJ}}{\partial w_{kj}} \,.$$

All the weights are independent variables, so that derivative will evaluate to zero unless $K = k$ and $J = j$, (this restriction is important, so keep it in mind as we move forward) so only one *k*-term survives the summation as does just a single *j*-term giving

$$\frac{\partial \Theta_J}{\partial w_{kj}} = \frac{\partial \Theta_j}{\partial w_{kj}} = a_k$$

which results in

$$\frac{\partial h_J}{\partial w_{kj}} = f'(\Theta_J) a_k$$

so

$$\frac{\partial \Theta_0}{\partial w_{kj}} = \sum_J w_{J0} f'(\Theta_J) a_k \,.$$

Here is were we need to remember that only one *J* term survives the summation because we required *K* = *k* and *J* = *j* to get nonzero terms in the derivative, so we simply have

$$\frac{\partial \Theta_0}{\partial w_{kj}} = w_{j0} f'(\Theta_j) a_k \,.$$

So we finally have

$$\frac{\partial E}{\partial w_{kj}} = -a_k \, \omega_0 f'(\Theta_0) w_{j0} f'(\Theta_j) \,.$$

Continuing with our theta notation and defining $\psi_0 = \omega_0 f'(\Theta_0)$ we have

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

If we now also define $\Omega_j = \psi_0 w_{j0}$ and $\Psi_j = \Omega_j f'(\Theta_j)$ we get out last change-in-error expression:

$$\frac{\partial E}{\partial w_{kj}} = -a_k \Psi_j \,.$$

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

$$h_j = f(\Theta_j) \qquad F_0 = f(\Theta_0)$$

$$\Theta_j = \sum_K a_K w_{Kj} \qquad \Theta_0 = \sum_J h_J w_{J0}$$

$$\Omega_j = \psi_0 w_{j0} \qquad \omega_0 = (T_0 - F_0)$$

$$\Psi_j = \Omega_j f'(\Theta_j) \qquad \psi_0 = \omega_0 f'(\Theta_0)$$

$$\frac{\partial E}{\partial w_{kj}} = -a_k \Psi_j \qquad \frac{\partial E}{\partial w_{j0}} = -h_j \psi_0$$

$$\Delta w_{kj} = -\lambda \frac{\partial E}{\partial w_{kj}} \qquad \Delta w_{j0} = -\lambda \frac{\partial E}{\partial w_{j0}}$$

You should save the change in the weights (Δw) in an array and then apply them in a final loop. So the final step is $w_{kj} += \Delta w_{kj}$ and $w_{j0} += \Delta w_{j0}$. A diagrammatic summary of the calculations makes the computational path a little clearer:

$$\begin{tikzcd}
n = 1\ (\mathit{input}) & n = 2 & n = 3\ (\mathit{output}) \\
k & kj & j0 \\
 & \Theta_j = \sum_K a_K w_{Kj} & \Theta_0 = \sum_J h_J w_{J0} \\
a_k \arrow[ru] & h_j = f(\Theta_j) \arrow[ru] & F_0 = f(\Theta_0) \\
 & \Omega_j = \psi_0 w_{j0} & \omega_0 = (T_0 - F_0) \\
 & \Psi_j = \Omega_j f'(\Theta_j) & \psi_0 = \omega_0 f'(\Theta_0) \\
 & \frac{\partial E}{\partial w_{kj}} = -a_k \Psi_j & \frac{\partial E}{\partial w_{j0}} = -h_j \psi_0 \\
 & \Delta w_{kj} = -\lambda \frac{\partial E}{\partial w_{kj}} & \Delta w_{j0} = -\lambda \frac{\partial E}{\partial w_{j0}} \\
\vec{w} += \Delta \vec{w} & &
\end{tikzcd}$$

Do NOT attempt to apply the changes in the weights immediately after they are calculated as there are dependencies that we have not yet explored. This is a design requirement at this point in the development cycle.

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 (excluding Python and Rust). 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 Δ*w* values for each weight in the network and apply them, iteratively, for each training instance 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. Lastly change the activation function to a sigmoid.

The Activation Function

To get *f'*(*x*) let us use a simple sigmoid so that $f(x)=\frac{1}{1+e^{-x}}$. In that case:

$$f'(x)=\frac{d}{dx}f(x)=\frac{d}{dx}\left(1+e^{-x}\right)^{-1}=-\left(1+e^{-x}\right)^{-2}e^{-x}=-f^2(x)\frac{d}{dx}e^{-x}=f^2(x)e^{-x},$$

but we can write $e^{-x}=\frac{1}{f(x)}-1$, so $\frac{d}{dx}f(x)=f^2(x)\left(\frac{1}{f(x)}-1\right)=f(x)\left(1-f(x)\right)$,

so we finally have $f'(x)=f(x)\left(1-f(x)\right)$ if $f(x)=\frac{1}{1+e^{-x}}$.