Let's review from the A-B-1 network. Consider some sort of network connectivity model with outputs F_i. Let our error function (also known as a 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. 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 by making it dependent on how fast things are changing.

You could also 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:

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

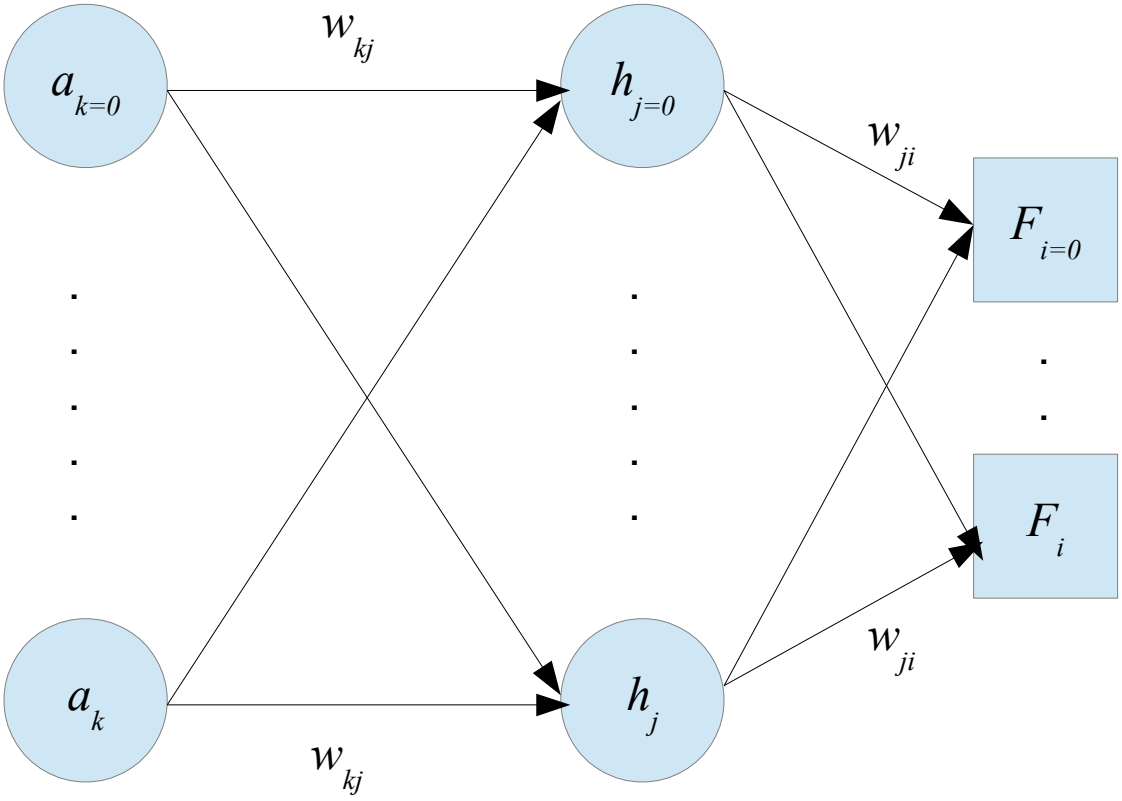
Or you could use the method of Davidon-Fletcher-Powell

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

The problem with these methods is that you must calculate the error function three times for each weight because of the derivative, which is computationally intensive. If you have a fast system, then it is not a problem. On the other hand, although they might be slow they can be used for ANY connectivity model.

For all these methods, at first it looks like we still need to make three determinations of the error function for each weight because we have the derivative to deal with, but a judicious choice of the connectivity model and activation rule along with a little grunt work shows how we can optimize the learning into a "back propagation" algorithm which significantly reduces the amount of computation. Back propagation is just an optimized version of steepest decent, so we will first need to implement steepest decent and then later optimize it.

We need to formalize our notation and our connectivity model. The initial use of our index letters was to help see the relations between the elements as they moved through the network. We now want to modify our notation to view things from right to left (yes, backwards through the layers) and start our index lettering on the output side:



With this notation the index variables i, j and k take on very specific meanings and shall not be used for any other purpose. The k-index is for the input layer, j is over the hidden activations and i is for the output layer. In this notation you will always see a_k, h_j and F_i. You will also only see specific pairings in a specific order: kj and ji.

What we want to know is how the error depends on the weights in each layer so we can determine how to modify the weights to minimize the error. We are going to limit our derivation to a network with only a single hidden activation layer (for now) which I shall call a two-layer network since it has two layers of weights. The hidden layer is fully connected to both its input and output activation layers.

We start with the expressions for the output values and an error function which is defined for only a single training case:

$$F_i = f\left(\sum_J h_J w_{Ji}\right)$$

and

$$E = \frac{1}{2}\sum_I \left(T_I - F_I\right)^2,$$

where $f(x)$ is the activation function. We should also note that by symmetry the values of the hidden activation nodes look a lot like the output activations in that

$$h_j = f\left(\sum_K a_K w_{Kj}\right).$$

You should notice that I am using capital letters for the index values that are being used inside of a summation. Lowercase index variables are not part of a summation and thus represent just a single arbitrary index value. This distinction will make things easier to follow as we move forward.

We want to define a few terms that will simplify the notation. Let

$$\Theta_i = \sum_J h_J w_{Ji} \text{ and } \Theta_j = \sum_K a_K w_{Kj}.$$

and

$$\omega_i = \left(T_i - F_i\right).$$

Our base relationships at the output layer then becomes

$$F_i = f\left(\Theta_i\right)$$

and

$$E = \frac{1}{2}\sum_I \omega_I^2.$$

First we want to see the effects of the error on the right-most set of weights coming out of the hidden layer (the *ji*-weights) which we saw can be expressed as

$$\Delta w_{ji} = -\lambda \frac{\partial E}{\partial w_{ji}} .$$

We can also see that

$$\frac{\partial E}{\partial w_{ji}} = \sum_I \omega_I \frac{\partial \omega_I}{\partial w_{ji}} .$$

Realizing that the T_I values are just constants in $\omega_i = (T_i - F_i)$, we can then easily write

$$\frac{\partial \omega_I}{\partial w_{ji}} = -\frac{\partial F_I}{\partial w_{ji}} .$$

Again we can continue to expand the right hand side to give

$$\frac{\partial F_I}{\partial w_{ji}} = f'(\Theta_I) \frac{\partial \Theta_I}{\partial w_{ji}} .$$

And that last derivative can be expressed as

$$\frac{\partial \Theta_I}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}} \sum_J h_J w_{JI} .$$

It is this expression where things start getting interesting. If you look at the diagram it is easy to see that the values of h_j have no dependency on the weights w_{ji}. That means we can rewrite the expression as

$$\frac{\partial}{\partial w_{ji}} \sum_J h_J w_{JI} = \sum_J h_J \frac{\partial}{\partial w_{ji}} w_{JI} .$$

All the weights are independent variables, so the only time $\frac{\partial}{\partial w_{ji}} w_{JI}$ will not be zero is when $J = j$ and $I = i$, so only one term in the summation survives, or more simply:

$$\frac{\partial \Theta_i}{\partial w_{ji}} = h_j .$$

We can therefore state (since we can now set $J = j$ and $I = i$)

$$\frac{\partial E}{\partial w_{ji}} = \omega_i \frac{\partial \omega_i}{\partial w_{ji}} = -\omega_i \frac{\partial F_i}{\partial w_{ji}} = -\omega_i f'(\Theta_i) \frac{\partial \Theta_i}{\partial w_{ji}} = -\omega_i f'(\Theta_i) h_j .$$

Let us define a new term

$$\psi_i = \omega_i f'(\Theta_i)$$

so we can finally write the change in the in the *ji*-weights as

$$\Delta w_{ji} = \lambda\, \psi_i h_j$$

where

$$\psi_i = \omega_i f'(\Theta_i),\ \omega_i = (T_i - F_i),\ \Theta_i = \sum_J h_J w_{Ji} \text{ and } F_i = f(\Theta_i).$$

Now let's move to the left most connectivity layer. We want to see the effects of the error on the first set of weights coming out of the initial input layer in our diagram (the *kj*-weights). Starting with

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

where

$$\frac{\partial E}{\partial w_{kj}} = \sum_I \omega_I \frac{\partial \omega_I}{\partial w_{kj}}.$$

Once again, the T_I values are just constants in ω_I so we have

$$\frac{\partial \omega_I}{\partial w_{kj}} = -\frac{\partial F_I}{\partial w_{kj}}.$$

Similar to what we did above we can write

$$\frac{\partial F_I}{\partial w_{kj}} = f'(\Theta_I)\frac{\partial \Theta_I}{\partial w_{kj}}$$

with

$$\frac{\partial \Theta_I}{\partial w_{kj}} = \frac{\partial}{\partial w_{kj}} \sum_J h_J w_{JI}.$$

And now things are not as simple as they were before. Looking at the diagram we see that the h_J values are dependent on the *kj*-weights. Fortunately, for our connectivity model, the *ji*-weights are independent of the *kj*-weights so we can write

$$\frac{\partial}{\partial w_{kj}} \sum_J h_J w_{JI} = \sum_J w_{JI} \frac{\partial}{\partial w_{kj}} h_J.$$

From above we have that

$$h_J = f(\Theta_J)$$

where

$$\Theta_J = \sum_K a_K w_{KJ},$$

so

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

where

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

We are now in the same situation we were before. The a_K values are not dependent on the *kj*-weights so they can be treated as constant, and all the *kj*-weights are independent variables, so the only time the derivative is not zero is when $K = k$ and $J = j$. It is important to keep this restriction in mind as we move forward since all the summations over *K* and *J* above will collapse down to single values for *k* and *j* as we unwind all the dependencies. Moving forward we thus have

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

We can therefore write

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

so

$$\frac{\partial \Theta_I}{\partial w_{kj}} = \sum_J w_{JI} a_k f'(\Theta_J)$$

but we have already determined that both *K* and *J* can only take on a single value, so we can drop the summation over *J* to give

$$\frac{\partial \Theta_I}{\partial w_{kj}} = w_{jI} a_k f'(\Theta_j)$$

and therefore

$$\frac{\partial F_I}{\partial w_{kj}} = f'(\Theta_I) w_{jI} a_k f'(\Theta_j)$$

and so

$$\frac{\partial \omega_I}{\partial w_{kj}} = -f'(\Theta_I) w_{jI} a_k f'(\Theta_j)$$

which gives

$$\frac{\partial E}{\partial w_{kj}} = -\sum_I \omega_I f'(\Theta_I) w_{jI} a_k f'(\Theta_j)$$

or using our definition for ψ_I

$$\frac{\partial E}{\partial w_{kj}} = -\sum_I \psi_I w_{jI} a_k f'(\Theta_j) .$$

Pulling out all the expressions from inside the summation that are not dependent on the *I*-index yields

$$\frac{\partial E}{\partial w_{kj}} = -a_k f'(\Theta_j) \sum_I \psi_I w_{jI} .$$

Now define

$$\Omega_j = \sum_I \psi_I w_{jI}$$

so

$$\frac{\partial E}{\partial w_{kj}} = -a_k f'(\Theta_j) \Omega_j$$

and also define

$$\Psi_j = f'(\Theta_j) \Omega_j$$

so

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

We now have the complete collection of expressions required to perform a steepest descent minimization of the error function.

$$\frac{\partial E}{\partial w_{kj}} = -a_k \Psi_j \text{ and } \frac{\partial E}{\partial w_{ji}} = -h_j \psi_i ,$$

where $h_j = f(\Theta_j)$, $\Psi_j = \Omega_j f'(\Theta_j)$, $\Theta_j = \sum_K a_K w_{Kj}$, $\Omega_j = \sum_I \psi_I w_{jI}$ and

$$\psi_i = \omega_i f'(\Theta_i), \ \omega_i = (T_i - F_i), \ F_i = f(\Theta_i), \ \Theta_i = \sum_J h_J w_{Ji}$$

You can now use steepest descent, $\Delta w_{\alpha\beta\gamma} = -\lambda \frac{\partial E}{\partial w_{\alpha\beta\gamma}}$, which resolves to

$$\Delta w_{kj} = \lambda a_k \Psi_j$$

and

$$\Delta w_{ji} = \lambda h_j \psi_i$$

for our two layer network with some learning factor λ. The last step, once we have all the changes in the weights, is to apply them. In other words

$w_{kj} \mathrel{+}= \Delta w_{kj}$ and $w_{ji} \mathrel{+}= \Delta w_{ji}$ for all values of i, j and k.

As before, do NOT attempt to apply the changes in the weights immediately after they are calculated as there are dependencies that we still have not yet explored. This is still a design requirement at this point in the development cycle. The summary of the data flow for executing, training, and applying the changes is shown below: