In the previous sets of notes we defined an error function (*aka* cost function) that was based on some expected outputs T_i compared to the actual outputs of the network F_i:

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

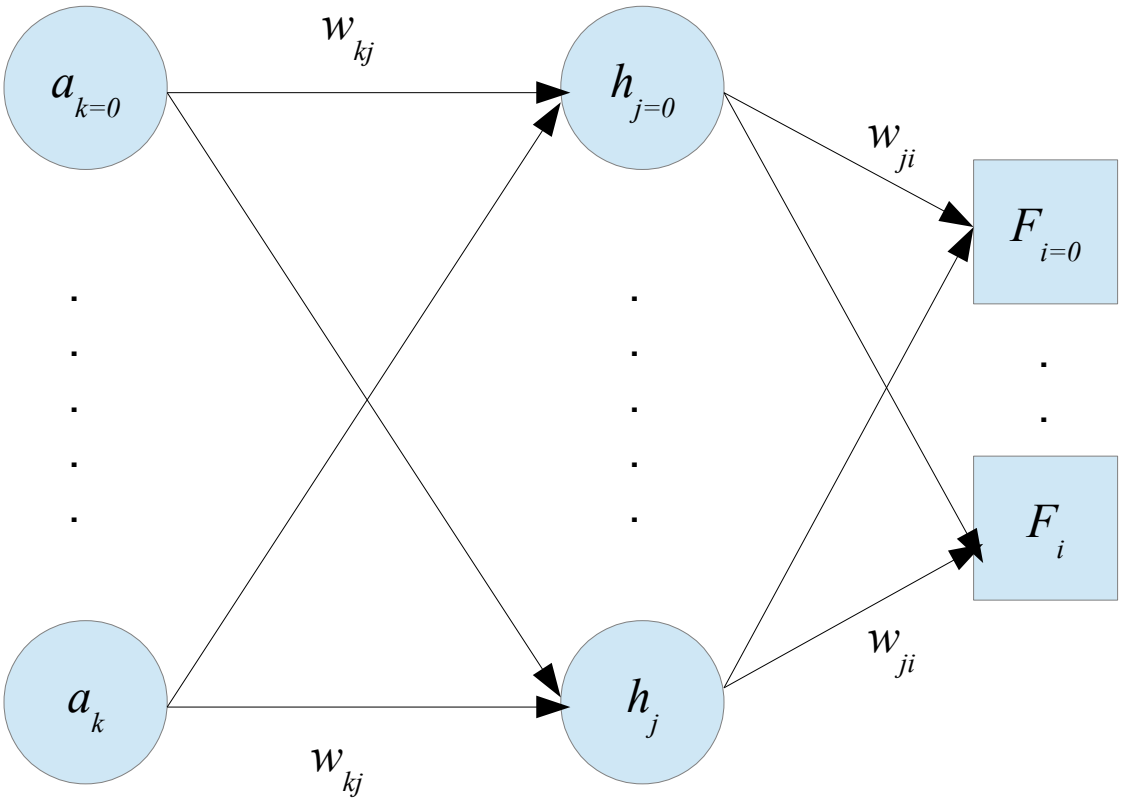
where

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

We then decided to use steepest descent, also known a gradient descent, to find the minimum of the error function using a "learning factor", which I am calling λ, to allow us to control how far things can move down hill in each step:

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

We defined a less generalized notation based on a two layer network as shown below.



We then derived the expressions needed to update the weights in both the connectivity layers. For the right most layer we have:

$$\Delta w_{ji} = \lambda h_j \psi_i \text{ where } \psi_i = \omega_i f'(\Theta_i),\ \omega_i = (T_i - F_i),\ F_i = f(\Theta_i) \text{ and } \Theta_i = \sum_j h_J w_{Ji}.$$

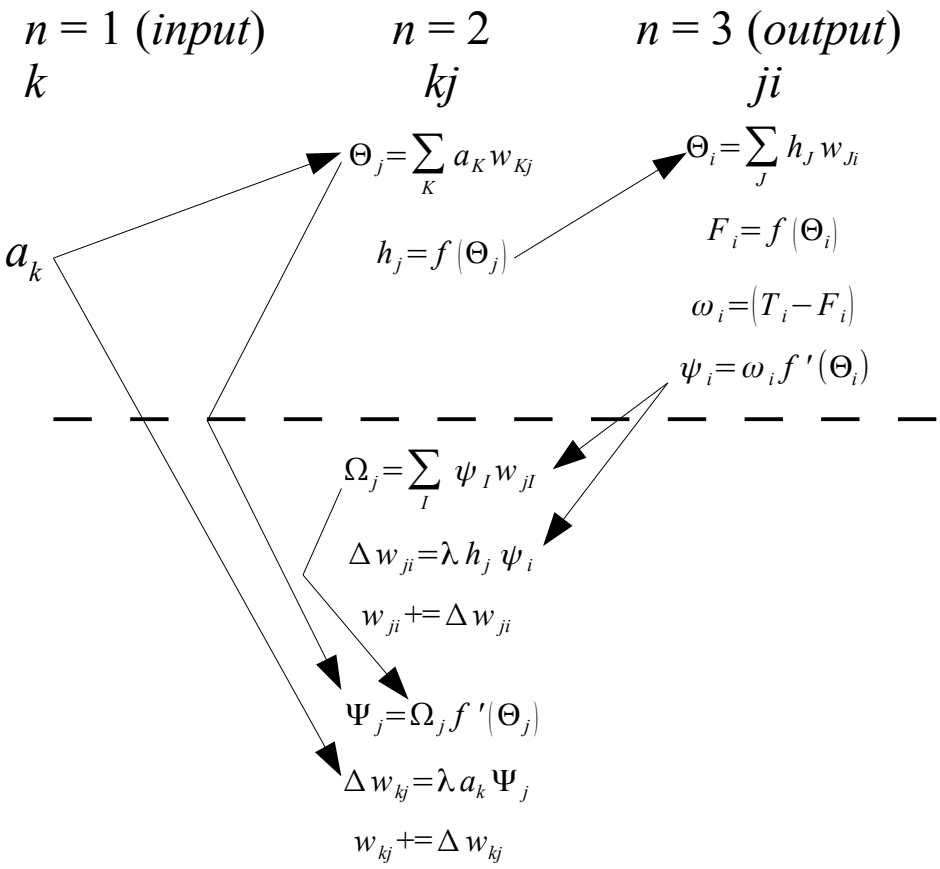
and for the left most layer:

$$\Delta w_{kj} = \lambda a_k \Psi_j \text{ where } \Psi_j = \Omega_j f'(\Theta_j),\ \Omega_j = \sum_I \psi_I w_{jI},\ \Theta_j = \sum_K a_K w_{Kj} \text{ and}$$
$$\psi_i = \omega_i f'(\Theta_i) \text{ where } \omega_i = (T_i - F_i),\ F_i = f(\Theta_i) \text{ and } \Theta_i = \sum_J h_J w_{Ji}.$$

We now have everything we need for a full back propagation implementation for a two layer network. The network is evaluated with the feed-forward calculations:

$$F_i = f(\Theta_i) \text{ where } \Theta_i = \sum_J h_J w_{Ji} \text{ and } h_j = f(\Theta_j) \text{ where } \Theta_j = \sum_K a_K w_{Kj}.$$

A diagram of the dependencies shows how the calculations are related and the required order of execution with the top being the forward evaluation for training and the bottom the back propagation adjustment of the weights:



Note that the change in weights on the right most layer are calculated and immediately applied in the layer to the left based on the calculation of the ψ_i values which are done when the network is evaluated, so there are no more dependencies on the weight values. You therefore only need one back propagation nested loop-construct over the *j*-index that encompasses everything.

Now we want to implement it. Remember that the point of the optimization is to minimize the number of loops needed to perform the training, so you will need to explicitly write out all the loops and then see how they can be combined, and the operations ordered, to minimize run time.

Evaluate the network: During the evaluation you will need to accumulate the items needed for the back propagation training (Θ and ψ values for example). Identify them for yourself. You need to remember that at some point you will simply be running the network with a predetermined set of weights and some arbitrary inputs (the training will be all done), so do NOT make execution of the network dependent on also training it. The pseudocode for <u>simply</u> evaluating the network and collecting the information needed for training looks <u>something</u> like :

```
for j = 1 to Nj    // kj layer
   Θj = 0          // Zero the Θj accumulator
   for k = 1 to Nk
      Θj += ak wkj
   next k
   hj = f(Θj)
next j

for i = 1 to Ni    //ji layer
   Θi = 0          // Zero the Θi accumulator
   for j = 1 to Nj
      Θi += hj wji
   next j
   Fi = f(Θi)
   ωi = Ti - Fi
   ψi = ωi f'(Θi)   // Output layer specific
next i
```

Train the network: The outer most loop will be over the training-set. The inner loops will again iterate over *i*, *j*, and *k*. Pay attention to the mathematics. As you calculate the Ω values, you will find that you can apply the change in weights on-the-fly once you are past the dependencies. **No extra loops or using duplicate copies of the weights or storing the delta-weights are needed**, so you will only need the equivalent of one set of nested loops for each connectivity layer. **There should be no conditionals in the execution or training algorithms**. You should write out the training in pseudo code similar to that for evaluating the network before you try to code it.