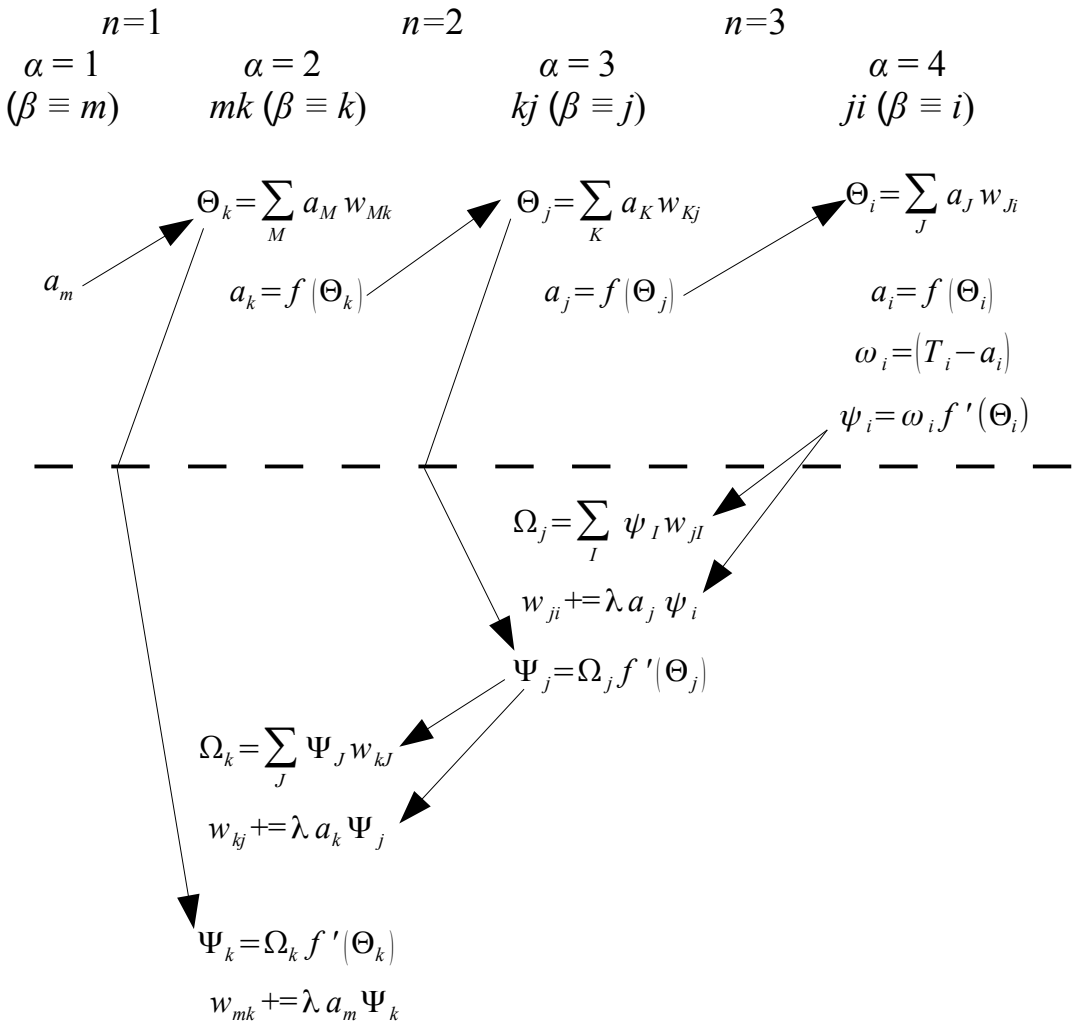
Consider the idea that all we have are different layers with activations *a*. Now consider a network with four activation layers. I have identified each layer by an index alpha (α) and have removed the special symbols for the input and output activations that have been used in the previous documentation. The various calculations that were derived for the three activation-layer network can be structured to more clearly show the interdependencies between the layers. I have also introduced a new index beta (β) that will soon be used to increment through the index values that are specific to a given layer. For the moment I am using the same convention as in the previous documents, that the index letter is associated with a specific layer.



Note that Ψ_j (which is tied to the output layer) is not the same as Ψ_k (which is tied to a hidden layer). You should notice that there will be an array of Θ values for every right-side activation layer, so there will be an array of values for every hidden layer and the output layer. These values can be collected when the activation/weight multiplications are being performed when the network is being evaluated. You will also need to maintain the activation values for every hidden layer.

Now for some details on the evaluation. The path down the network to evaluate for training is

```
for k = 1 to Nk    // mk layer
   Θk = 0          // Zero the Θk accumulator
   for m = 1 to Nm
      Θk += am * wmk
   next m
   ak = f(Θk)
next k

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

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

It should be clear from the diagram (and the pseudo code) that we can make a generalized set of constructs to hold the various elements that are required to be propagated down and back up the network. The only layer that has any special elements is the bottom (the output layer) since it is where the training models come in. We need to loop over all the layers starting from the right-most activation layer, which has the outputs. Any variable that previously had a single index value (which implied the layer) will require two and the weights three. The first index is the layer, index α, and the second index is the specific element within that layer, call this index β, so we have activations $a_{\alpha\beta}$. The beta index is what was previously represented by the i, j, k, and m indices that were also associated with a specific activation layer. The idea here is to move from having a, h and F arrays to having only a single two dimensional activation array as well as recognizing that we need a single two dimensional theta array and a single three dimensional weights array. Using this notation, the above loop becomes:

```
for k = 1 to Nk // mk layer
   Θ2k = 0
   for m = 1 to Nm
      Θ2k += a1m w1mk
   next m
   a2k = f(Θ2k)
next k

for j = 1 to Nj // kj layer
   Θ3j = 0
   for k = 1 to Nk
      Θ3j += a2k w2kj
   next k
   a3j = f(Θ3j)
next j

for i = 1 to Ni // ji layer
   Θ4i = 0
   for j = 1 to Nj
      Θ4i += h3j w3ji
   next j
   a4i = f(Θ4i)
   ωi = Ti - a4i
   ψi = ωi f'(Θ4i)
next i
```

where $\alpha = 4$ is associated with the *i* index, $\alpha = 3$ with the *j* index, and so on. Look at the associations of the layer index and the element index within that layer and you find that 1*m*, 2*k*, 3*j*, and 4*i* now identify each of the elements in a specific layer. Note that the N values in the above pseudocode represent the highest index and not necessarily the number of elements (zero indexing *vs.* one indexing).