In the above derivation we have implicitly assumed that each hidden or output unit in the network has the same activation function $h(\cdot)$. However, the derivation is easily generalized to allow different units to have individual activation functions, simply by keeping track of which form of $h(\cdot)$ goes with which unit.

8.1.3 A simple example

The above derivation of the backpropagation procedure allowed for general forms for the error function, the activation functions, and the network topology. To illustrate the application of this algorithm, we consider a two-layer network of the form illustrated in Figure 6.9, together with a sum-of-squares error. The output units have linear activation functions, so that $y_k = a_k$, and the hidden units have sigmoidal activation functions given by

$$h(a) \equiv \tanh(a) \tag{8.15}$$

where $\tanh(a)$ is defined by (6.14). A useful feature of this function is that its derivative can be expressed in a particularly simple form:

$$h'(a) = 1 - h(a)^2. \tag{8.16}$$

We also consider a sum-of-squares error function, so that for data point n the error is given by

$$E_n = \frac{1}{2}\sum_{k=1}^{K}(y_k - t_k)^2 \tag{8.17}$$

where y_k is the activation of output unit k, and t_k is the corresponding target value for a particular input vector $\mathbf{x}_n$.

For each data point in the training set in turn, we first perform a forward propagation using

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i \tag{8.18}$$

$$z_j = \tanh(a_j) \tag{8.19}$$

$$y_k = \sum_{j=0}^{M} w_{kj}^{(2)} z_j \tag{8.20}$$

where D is the dimensionality of the input vector $\mathbf{x}$ and M is the total number of hidden units. Also we have used $x_0 = z_0 = 1$ to allow bias parameters to be included in the weights. Next we compute the δ's for each output unit using

$$\delta_k = y_k - t_k. \tag{8.21}$$

Then, we backpropagate these errors to obtain δ's for the hidden units using

$$\delta_j = (1 - z_j^2)\sum_{k=1}^{K} w_{kj}^{(2)} \delta_k, \tag{8.22}$$