the other eigenvalues are 0, therefore, the magnitude of $\mathbf{v}^{(m)}$ monotonically decreases if $0 < \lambda < 2$. If there exists no consistent solution, then a minimum-squared-error solution is the best possible alternative, which incurs a residual vector

$$\mathbf{r} = [r^{(1)},\ r^{(2)},\ \cdots,\ r^{(M)}]^T$$

and $\mathbf{A}\mathbf{w}^* + \mathbf{r} = \mathbf{t}$. Now Eq. 5.8 can be rewritten as

$$\begin{aligned}
\mathbf{v}^{(m+1)} &= \mathbf{w}^* - \mathbf{w}^{(m+1)} \\
&= \mathbf{w}^* - \mathbf{w}^{(m)} - \lambda \left(t^{(m)} - \mathbf{z}^{(m)^T}\mathbf{w}^{(m)}\right) \frac{\mathbf{z}^{(m)}}{|\mathbf{z}^{(m)}|^2} \\
&= \mathbf{v}^{(m)} - \lambda \left(\mathbf{z}^{(m)^T}\mathbf{w}^* - \mathbf{z}^{(m)^T}\ \mathbf{w}^{(m)}\right) \frac{\mathbf{z}^{(m)}}{|\mathbf{z}^{(m)}|^2} - \lambda\, r^{(m)}\, \frac{\mathbf{z}^{(m)}}{|\mathbf{z}^{(m)}|^2} \\
&= \mathbf{v}^{(m)} - \lambda\, \mathbf{z}^{(m)^T}\ \mathbf{v}^{(m)}\, \frac{\mathbf{z}^{(m)}}{|\mathbf{z}^{(m)}|^2} - \lambda\, r^{(m)}\, \frac{\mathbf{z}^{(m)}}{|\mathbf{z}^{(m)}|^2} \\
&= \left(\mathbf{I} - \lambda\, \frac{\mathbf{z}^{(m)}\mathbf{z}^{(m)^T}}{|\mathbf{z}^{(m)}|^2}\right)\mathbf{v}^{(m)} - \lambda\, r^{(m)}\, \frac{\mathbf{z}^{(m)}}{|\mathbf{z}^{(m)}|^2}
\end{aligned} \tag{5.9}$$

The estimation residual (the second term on the right-hand side) prevents the weights from reaching the desired solution $\mathbf{w}^*$. To reduce its effect, a narrower range of λ, $0 < \lambda \leq 1$, is preferred. Moreover, in the final converging stage, λ must be set to be gradually decreasing, for example, inversely proportional to m. For comparison, a more conservative convergence analysis as discussed in Problem 5.4 would yield a slower learning rate.

5.3 Nonlinear Multilayer Back-Propagation Networks

It can be shown that [196] single-layer nets based on linear model functions have very limited classification and approximation capabilities. More precisely,

1. Linear approximation networks are too restrictive and nonlinear approximation networks offer much greater capacity.

2. In order to enhance the approximation capabilities, it is critical to expand a single-layer structure to a multilayer network.

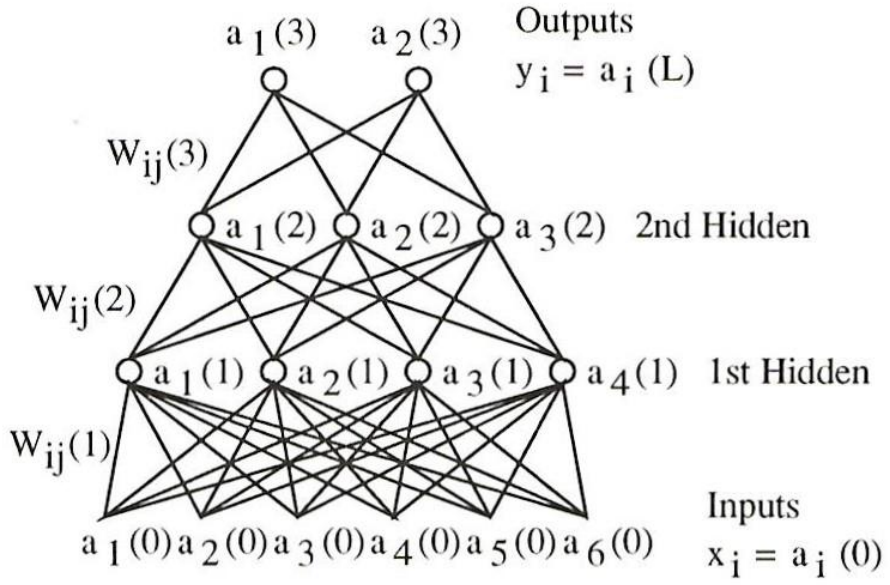

Figure 5.3: The multiple-layer network shown here is a net with three weight layers. The weight layers support linear transformations. The neuron layers perform (local) nonlinear transformations.

The number of nonlinear layers of a multilayer network is defined to be the number of weight layers, instead of neuron layers. For example, the network depicted in Figure 5.3 is called a three-layer network because it has *three* weight layers. In linear systems, there is no real benefit to cascading multiple layers of *linear networks.* The equivalent weight matrix for the total system is simply the product of the weight matrices of different layers.

The situation is quite different if nonlinear *hidden neuron units* are inserted between the input and output layers. In this case, it seems natural to assume that the more layers used, the greater power the network possesses. However, it is not the case in practice. An excessive number of layers often proves to be counterproductive. It may cause slower convergence in the back-propagation learning. Two possible reasons are that the error signals may be numerically degraded when propagating across too many layers and that extra layers tend to create additional local minima. Thus, it is essential to identify the proper number of layers. Generally speaking, two-layer network should be adequate as universal approximators of any nonlinear functions, cf. Theorems 5.1 and 5.2. It was further demonstrated in [174] that a three-layer network suffices to separate any (convex or nonconvex) polyhedral decision region from its background. In summary, two or three layers should be adequate for most applications.

5.3.1 Back-Propagation Algorithm

The back-propagation (BP) algorithm offers an effective approach to the computation of the gradients. This can be applied to any optimization formulation (i.e., any type of energy function) as well as the DBNN formulation introduced in Chapter 4.

A linear basis function (LBF) multilayer network is characterized by the following dynamic equations

$$u_i(l) = \sum_{j=1}^{N_{l-1}} w_{ij}(l) a_j(l-1) + \theta_i(l) \tag{5.10}$$

$$a_i(l) = f(u_i(l)) \quad 1 \le i \le N_l; \quad 1 \le l \le L \tag{5.11}$$

where the input units are represented by $x_i \equiv a_i(0)$, the output units by $y_i \equiv a_i(L)$, and where L is the number of layers. The activation function is very often a *sigmoid function:*

$$f(u_i) = \frac{1}{1 + e^{-u_i/\sigma}}$$

Other dynamic equations are also of possible interest. For example, it is very popular to use Gaussian activation function on radial basis functions (RBF).

The back-propagation algorithm, independently proposed by Werbos [301], Parker [216], and Rumelhart [246], offers an efficient computational speed-up for training multilayer networks. The objective is to train the weights w_{ij} so as to minimize E. The basic gradient-type learning formula is

$$w_{ij}^{(m+1)}(l) = w_{ij}^{(m)}(l) + \Delta w_{ij}^{(m)}(l) \tag{5.12}$$

with the mth training pattern, $\mathbf{a}^{(m)}(0)$, and its corresponding teacher $\mathbf{t}^{(m)}$, $m = 1, 2, \cdots, M$, presented. The derivation of the BP algorithm follows a chain-rule technique:

$$\Delta w_{ij}^{(m)}(l) = -\eta \frac{\partial E}{\partial w_{ij}^{(m)}(l)} \tag{5.13}$$

$$= -\eta \frac{\partial E}{\partial a_i^{(m)}(l)} \frac{\partial a_i^{(m)}(l)}{\partial w_{ij}^{(m)}(l)}$$

$$= \eta \, \delta_i^{(m)}(l) \, f'(u_i^{(m)}(l)) \, a_j^{(m)}(l-1) \tag{5.14}$$

where the error signal $\delta_i^{(m)}(l)$ is defined as

$$\delta_i^{(m)}(l) \equiv -\frac{\partial E}{\partial a_i^{(m)}(l)}$$

Back-Propagation Rule for Approximation-Based Networks The aforementioned algorithm can be applied to training approximation-based networks. In this case, the objective is to train the weights w_{ij} and the thresholds θ_i, so as to minimize the least-squares-error between the teacher and the actual response [246]. That is,

$$E = \frac{1}{2} \sum_{m=1}^{M} \sum_{i=1}^{N} [t_i^{(m)} - a_i^{(m)}(L)]^2 \tag{5.15}$$

where M is the number of training patterns and N is the dimension of the output space. The back-propagation algorithm can be summarized in two steps:

I. The error signal $\delta_i^{(m)}(l)$ can be obtained recursively by back propagation:

- **Initial (Top) Layer** For the recursion, the initial value (of the top layer), $\delta_i^{(m)}(L)$, can be easily obtained as follows:

$$\begin{aligned} \delta_i^{(m)}(L) &\equiv -\frac{\partial E}{\partial a_i^{(m)}(L)} \\ &= t_i^{(m)} - a_i^{(m)}(L) \end{aligned} \tag{5.16}$$

  For an energy function other than the LSE, the initial condition can be similarly derived.

- **Recursive Formula** The general BP recursive formula for the error signal $\delta_i^{(m)}(l)$ can be derived as follows:

$$\begin{aligned} \delta_i^{(m)}(l) &\equiv -\frac{\partial E}{\partial a_i^{(m)}(l)} \\ &= -\sum_{j=1}^{N_{l+1}} \frac{\partial E}{\partial u_j^{(m)}(l+1)} \frac{\partial u_j^{(m)}(l+1)}{\partial a_i^{(m)}(l)} \end{aligned} \tag{5.17}$$

  in the sequence of $l = L-1, \ldots, 1$.

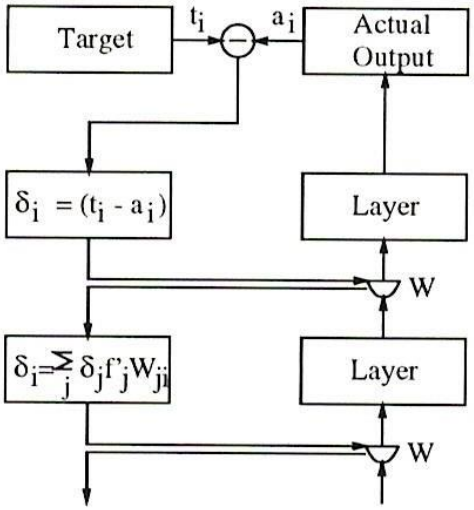

Figure 5.4: The schematic diagram for the back-propagation process.

From Eq. 5.11, it follows that

$$\delta_i^{(m)}(l) = \sum_{j=1}^{N_{l+1}} \delta_j^{(m)}(l+1)\ f'(u_j^{(m)}(l+1))\ w_{ji}^{(m)}(l+1) \quad (5.18)$$

II. Based on Equations 5.12 and 5.14, the synaptic weights between the (lth and $(l-1)$th) layers can be updated recursively (in order of $l = L, L-1, \ldots, 1$)

$$w_{ij}^{(m+1)}(l) = w_{ij}^{(m)}(l) + \eta \delta_i^{(m)}(l)\ f'(u_i^{(m)}(l))\ a_j^{(m)}(l-1) \quad (5.19)$$

The recursive formula is the key to back-propagation learning. It allows the error signal of a lower layer $\delta_j^{(m)}(l)$ to be computed as a linear combination of the error signal of the upper layer $\delta_j^{(m)}(l+1)$. In this manner, the error signals $\delta_j^{(m)}(\cdot)$ are back propagated through all the layers from the top down. This also implies that the influences from an upper layer to a lower layer (and vice versa) can only be effected via the error signals of the intermediate layer.

5.3.2 Numerical Back-Propagation Methods

The goal of training is to minimize the sum-of-squares-error energy function

$$E = \sum_{m=1}^{M} E^{(m)} \quad (5.20)$$

where

$$E^{(m)} = \frac{1}{2}\sum_{i=1}^{N_L}\left\{t_i^{(m)} - y_i^{(m)}\right\}^2 \tag{5.21}$$

We need to define *iterations* and *sweeps*. An *iteration* involves a single training datum to the system. A *sweep* covers the presentation of an entire block of training data. In most training practices, multiple sweeps are used and the training patterns are repeatedly presented in a cyclic manner. Assume for convenience that the number of input units, the number of hidden units, and the number of output units are all equal to N. For the two-layer case, the conventional gradient-descent BP requires $\mathcal{O}(5N^2)$ per iteration. For three-layer networks, the conventional BP method requires $\mathcal{O}(7N^2)$ operations per iteration. See Problems 5.5 and 5.6.

The approximation formulation can be viewed as a numerical optimization problem. The numerical method adopted has a direct effect on the performance of the BP algorithm. The rich literature on the study of numerical methods for optimization should be fully explored and utilized [182]. The numerical performance of the BP method depends on the following key factors.

- **Frequency of Update: Data-Adaptive vs. Block-Adaptive:** The choice is between block-adaptive methods (one update per block) and data-adaptive methods (one update per datum). The data-adaptive methods update the weights at each iteration, as opposed to the block methods, which execute the update upon the completion of each sweep. The back-propagation learning algorithm can be executed in either mode. The two approaches have exhibited significantly different numerical performances. Block-adaptive methods are known to be more robust since the training step is averaged over all the training patterns. In contrast, data-adaptive methods can be appealing for some on-line adaptation applications. They are more sensitive to the noise effect on individual patterns. Block methods are recommended for applications where real-time learning is not necessary.

- **Direction of Update: First-Order vs. Second-Order:** For both the first-order and second-order methods, the BP algorithm's sole role is as an effective tool for computing the gradients. Second-order gradients are numerically sensitive to compute. Therefore, data-adaptive methods are usually based on first-order gradients. Block-adaptive methods