Chapter 6:
Multilayer Neural Networks

- Feedforward Operation
- Backpropagation Algorithm

Introduction

- Goal: Classify objects by learning nonlinearity
  - There are many problems for which linear discriminants are insufficient for minimum error
  - In previous methods, the central difficulty was the choice of the appropriate nonlinear discriminant functions
  - A “brute” approach might be to select a complete basis set such as all polynomials; such a classifier would require too many parameters to be determined from a limited number of training samples

- There is no automatic method for determining the nonlinearities when no information is provided to the classifier

- In using the multilayer Neural Networks, the form of the nonlinearity is learned from the training data

Feedforward Operation and Classification

- A three-layer neural network consists of an input layer, a hidden layer and an output layer interconnected by modifiable weights represented by links between layers

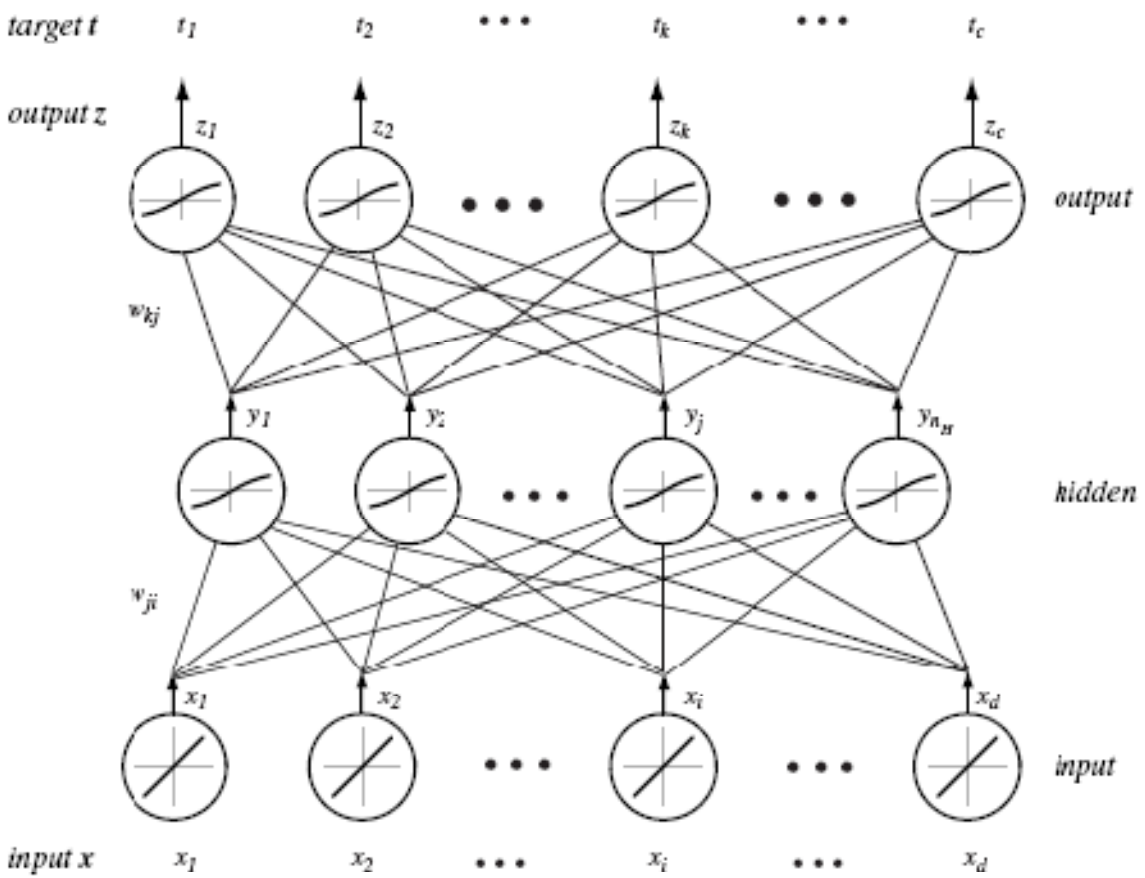


FIGURE 6.4. A d-n_H-c fully connected three-layer network and the notation we shall use. During feedforward operation, a d-dimensional input pattern $\mathbf{x}$ is presented to the input layer; each input unit then emits its corresponding component x_i. Each of the n_H hidden units computes its net activation, net_j, as the inner product of the input layer signals with weights w_{ji} at the hidden unit. The hidden unit emits $y_j = f(net_j)$, where $f(\cdot)$ is the nonlinear activation function, shown here as a sigmoid. Each of the c output units functions in the same manner as the hidden units do, computing net_k as the inner product of the hidden unit signals and weights at the output unit. The final signals emitted by the network, $z_k = f(net_k)$, are used as discriminant functions for classification. During network training, these output signals are compared with a teaching or target vector $\mathbf{t}$, and any difference is used in training the weights throughout the network. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

- A single “bias unit” is connected to each unit other than the input units

- Net activation: $$net_j = \sum_{i=1}^{d} x_i w_{ji} + w_{j0} = \sum_{i=0}^{d} x_i w_{ji} \equiv w_j^t . x,$$

  where the subscript i indexes units in the input layer, *j* in the hidden; w_{ji} denotes the input-to-hidden layer weights at the hidden unit *j*. (In neurobiology, such weights or connections are called “synapses”)

- Each hidden unit emits an output that is a nonlinear function of its activation, that is: $y_j = f(net_j)$

A simple threshold function

$$f(net) = sgn(net) \begin{cases} 1 & \text{if } net \geq 0 \\ -1 & \text{if } net < 0 \end{cases}$$

- The function *f(.)* is also called the activation function or "nonlinearity" of a unit. There are more general activation functions with desirables properties

- Each output unit similarly computes its net activation based on the hidden unit signals as:

$$net_k = \sum_{j=1}^{n_H} y_j w_{kj} + w_{k0} = \sum_{j=0}^{n_H} y_j w_{kj} = \mathrm{w}_k^t . \mathrm{y},$$

where the subscript *k* indexes units in the output layer and n_H denotes the number of hidden units

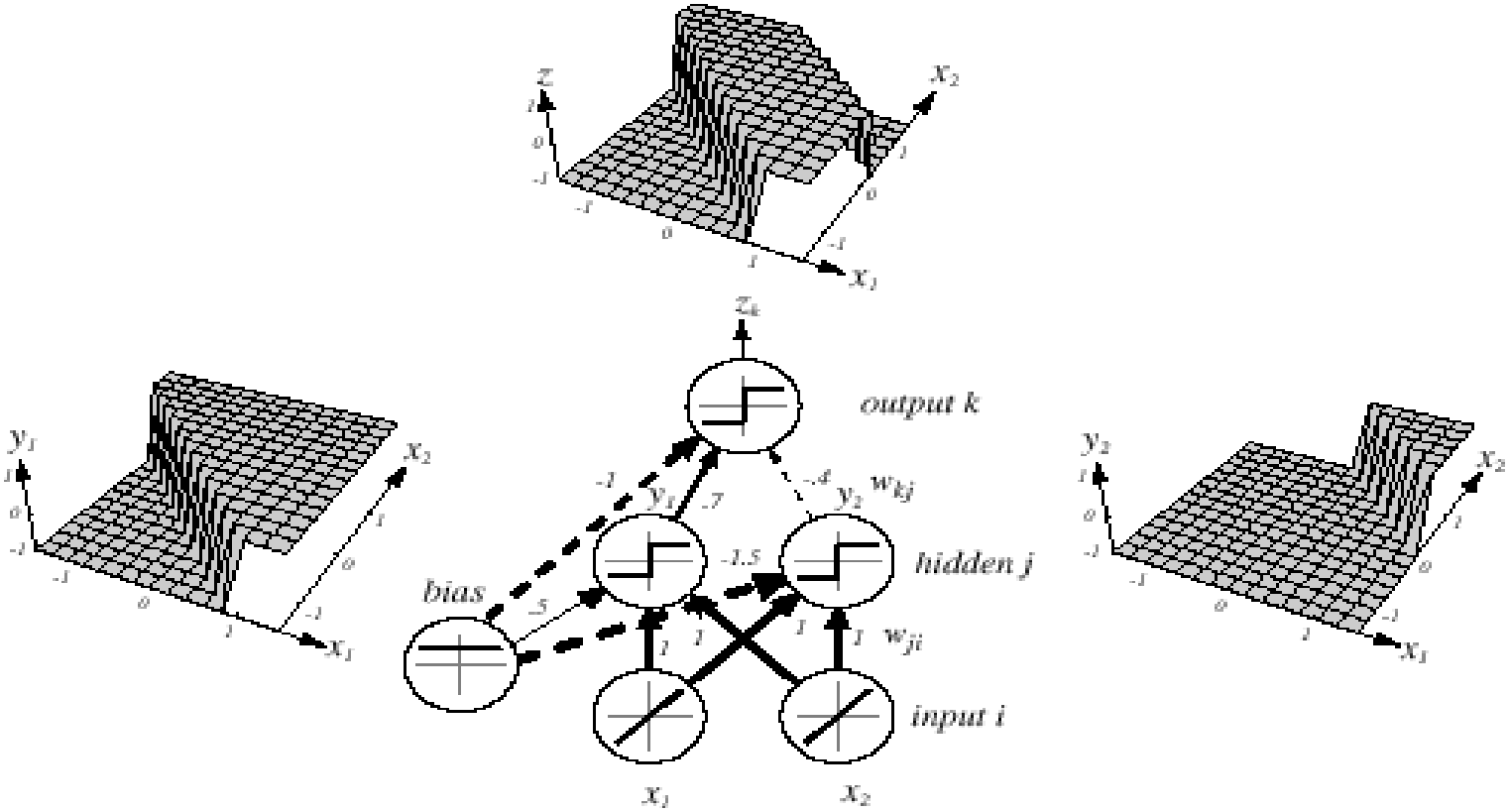
- More than one output are referred z_k. An output unit computes the nonlinear function of its net, emitting

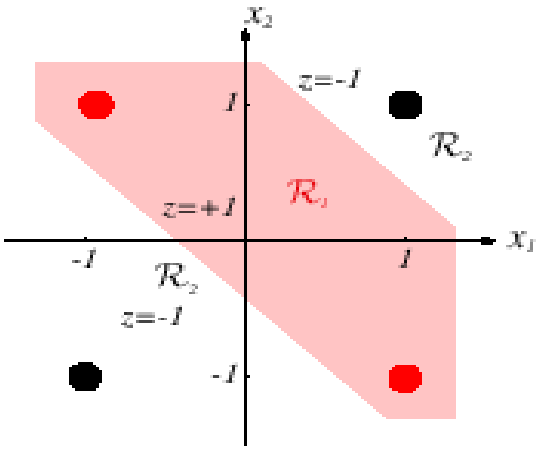
$$z_k = f(net_k)$$

- In the case of c outputs (classes), we can view the network as computing c discriminant functions

  $z_k = g_k(x)$ and classify the input x according to the largest discriminant function $g_k(x)$ $\forall\ k = 1, \ldots, c$

- The three-layer network with the weights listed in Fig. 6.1 solves the XOR problem


z
x2
x1
y1
y2
zk
output k
bias
hidden j
input i
wkj
wji
-1
.7
-.4
-1.5
.5
1
1
1
1
1
1
1
0
-1

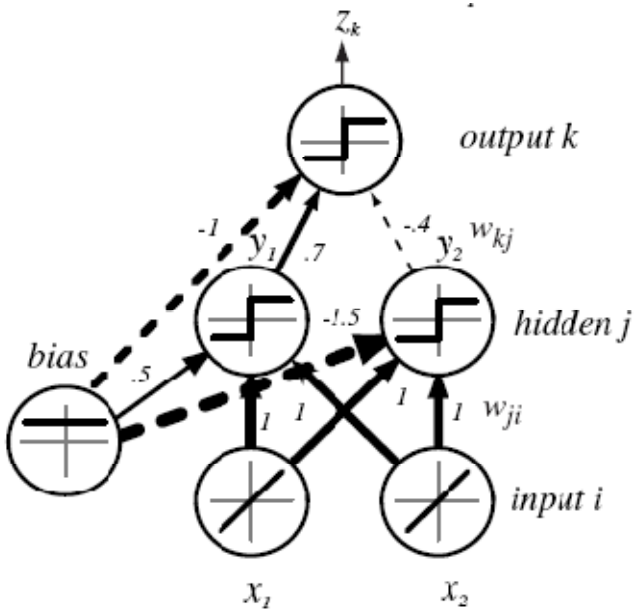

x2
x1
z=-1
z=+1
z=-1
1
-1
1
-1

- The hidden unit y_1 computes the LDF:

$$x_1 + x_2 + 0.5 \begin{cases} \geq 0 \Rightarrow y_1 = +1 \\ < 0 \Rightarrow y_1 = -1 \end{cases} \quad (OR)$$

- The hidden unit y_2 computes the LDF:

$$x_1 + x_2 - 1.5 \begin{cases} \geq 0 \Rightarrow y_2 = +1 \\ < 0 \Rightarrow y_2 = -1 \end{cases} \quad (AND)$$



- The final output unit emits z

$$0.7y_1 - 0.4y_2 - 1 \begin{cases} \geq 0 \Rightarrow z = +1 \\ < 0 \Rightarrow z = -1 \end{cases} \quad (y_1 \text{ AND } -y_2)$$

$z = y_1$ and not $y_2 = (x_1 \text{ or } x_2)$ and not $(x_1 \text{ and } x_2) = x_1 \text{ XOR } x_2$

which provides the nonlinear decision of Fig. 6.1

- General Feedforward Operation – case of c output units

$$g_k(x) \equiv z_k = f\left(\sum_{j=1}^{n_H} w_{kj} f\left(\sum_{i=1}^{d} w_{ji} x_i + w_{j0}\right) + w_{k0}\right) \qquad (1)$$

$$(k = 1,\ldots,c)$$

- Hidden units enable us to express more complicated nonlinear functions and thus extend the classification
- The activation function does not have to be a sign function, it is often required to be continuous and differentiable
- We can allow the activation in the output layer to be different from the activation function in the hidden layer or have different activation for each individual unit
- We assume for now that all activation functions to be identical

q Expressive Power of multi-layer Networks

Question: Can every decision be implemented by a three-layer network described by equation (1) ?

Answer: Yes (due to A. Kolmogorov)

"Any continuous function from input to output can be implemented in a three-layer net, given sufficient number of hidden units n_H, proper nonlinearities, and weights."

$$g(x) = \sum_{j=1}^{2n+1} \Xi_j \left(\Sigma y_{ij}(x_i) \right) \quad \forall \mathrm{x} \in \mathrm{I}^{\mathrm{n}} (I = [0,1]; n \geq 2)$$

for properly chosen functions X_j and y_{ij}

- Each of the *2n+1* hidden units X_j takes as input a sum of *d* nonlinear functions, one for each input feature x_i
- Each hidden unit emits a nonlinear function X_j of its total input
- The output unit emits the sum of the contributions of the hidden units

Unfortunately: Kolmogorov's theorem tells us very little about how to find the nonlinear functions based on data; this is the central problem in network-based pattern recognition

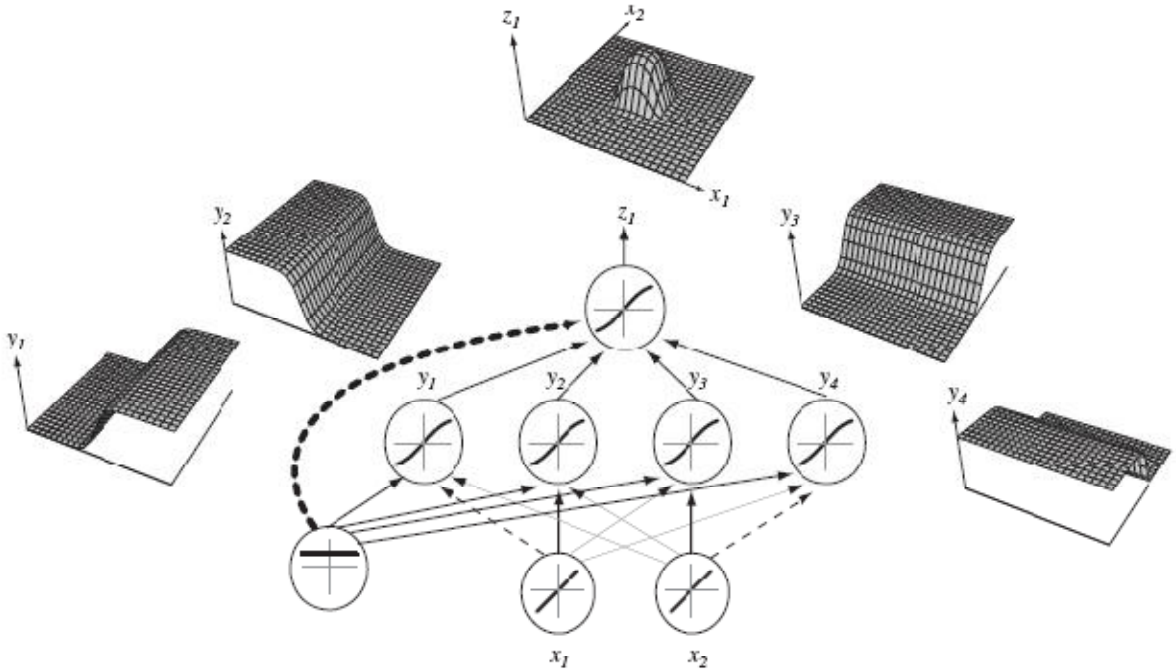


FIGURE 6.2. A 2-4-1 network (with bias) along with the response functions at different units; each hidden output unit has sigmoidal activation function $f(\cdot)$. In the case shown, the hidden unit outputs are paired in opposition thereby producing a "bump" at the output unit. Given a sufficiently large number of hidden units, any continuous function from input to output can be approximated arbitrarily well by such a network. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Backpropagation Algorithm

- Any function from input to output can be implemented as a three-layer neural network

- These results are of greater theoretical interest than practical, since the construction of such a network requires the nonlinear functions and the weight values which are unknown!

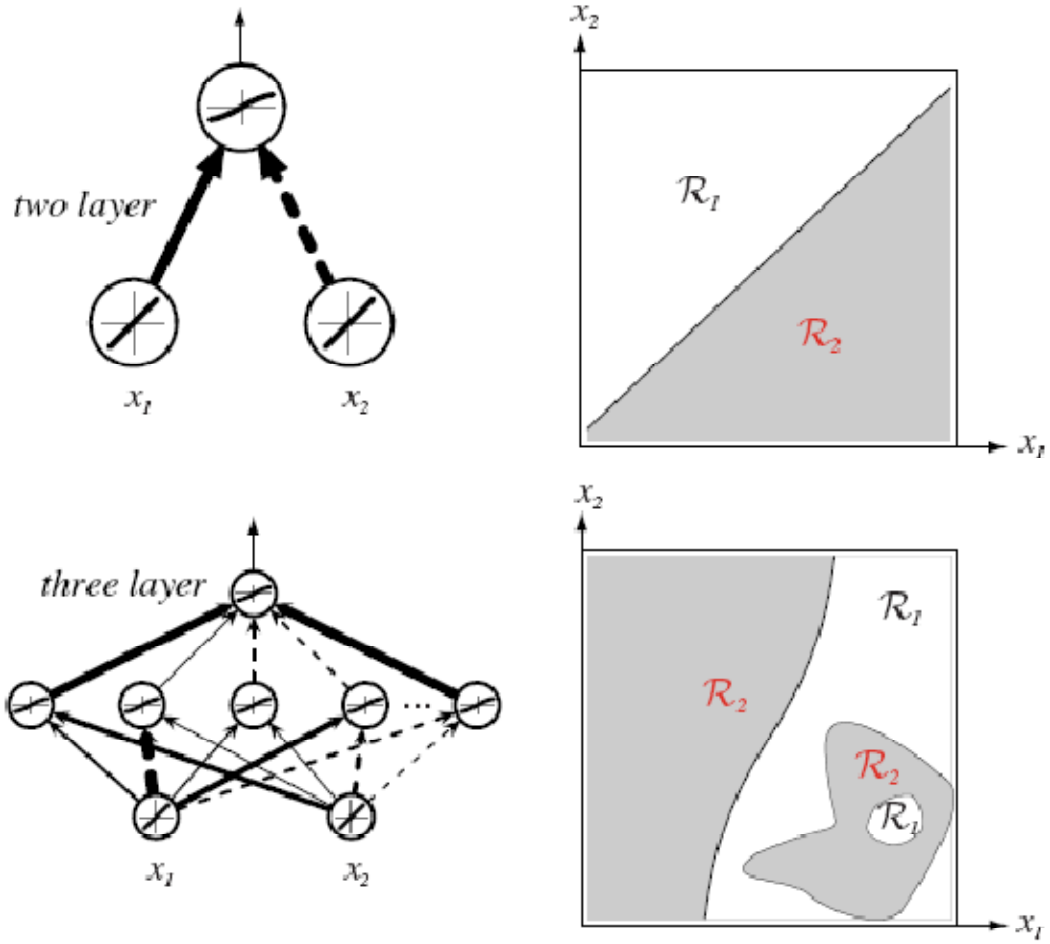


FIGURE 6.3. Whereas a two-layer network classifier can only implement a linear decision boundary, given an adequate number of hidden units, three-, four- and higher-layer networks can implement arbitrary decision boundaries. The decision regions need not be convex or simply connected. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

- Our goal now is to set the interconnexion weights based on the training patterns and the desired outputs

- In a three-layer network, it is a straightforward matter to understand how the output, and thus the error, depend on the hidden-to-output layer weights

- The power of backpropagation is that it enables us to compute an effective error for each hidden unit, and thus derive a learning rule for the input-to-hidden weights, this is known as:

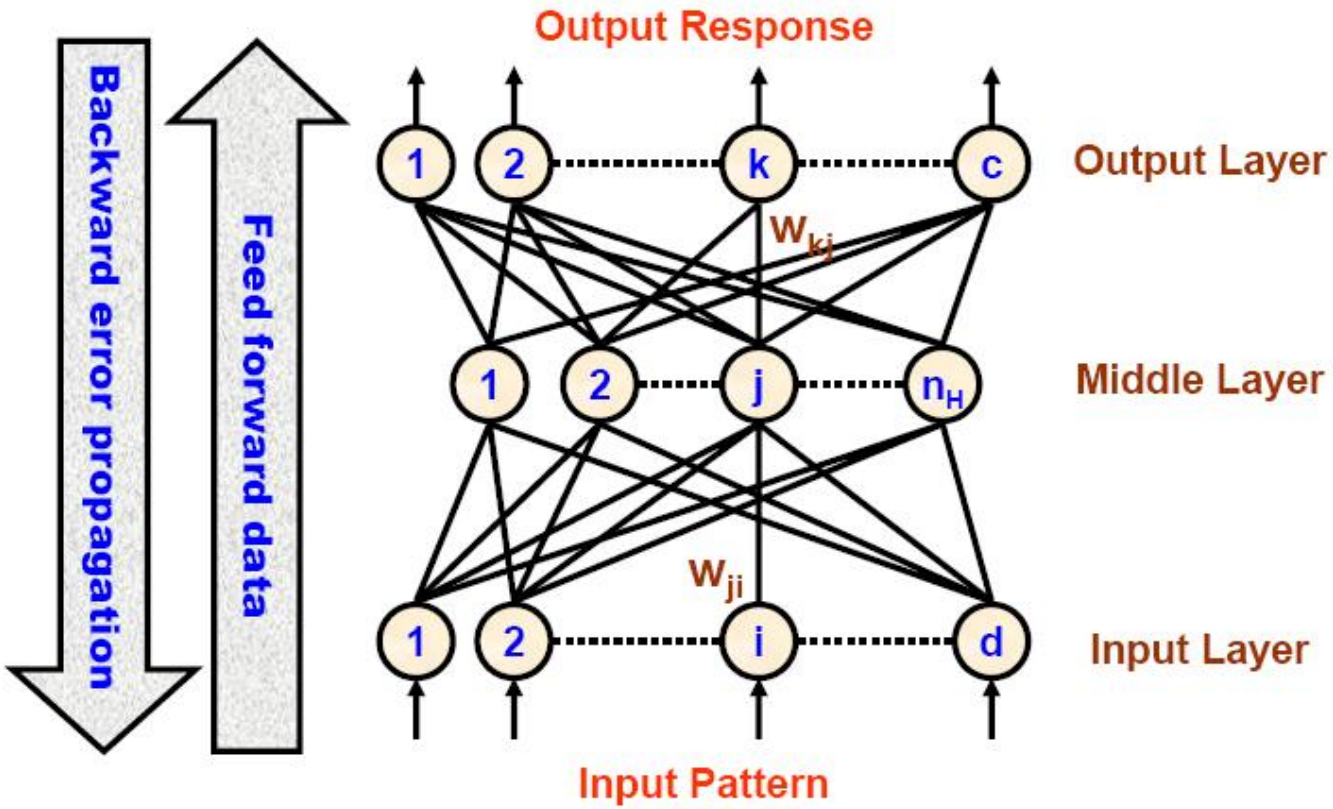
The credit assignment problem

- Network have two modes of operation:
  - Feedforward

    The feedforward operations consists of presenting a pattern to the input units and passing (or feeding) the signals through the network in order to get outputs units
  - Learning

    The supervised learning consists of presenting an input pattern and modifying the network parameters (weights) to reduce distances between the computed output and the desired output

BACK-PROPAGATION NETWORK



- Network Learning
  - Let tk be the k-th target (or desired) output and zk be the k-th computed output with $k = 1, \ldots, c$ and w represents all the weights of the network
  - The training error: $J(w) = \frac{1}{2} \sum_{k=1}^{c} (t_k - z_k)^2 = \frac{1}{2} \|t - z\|^2$
  - The backpropagation learning rule is based on gradient descent
    - The weights are initialized with pseudo-random values and are changed in a direction that will reduce the error:

$$\Delta w = -b \frac{\partial J}{\partial w}$$

where b is **the learning rate** which indicates the relative size of the change in weights

$$w(m+1) = w(m) + \Delta w(m)$$

where m is the m-th pattern presented

- Error on the hidden–to-output weights

$$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial net_k} \cdot \frac{\partial net_k}{\partial w_{kj}} = -\delta_k \frac{\partial net_k}{\partial w_{kj}}$$

where the sensitivity of unit k is defined as: $\delta_k = -\frac{\partial J}{\partial net_k}$

and describes how the overall error changes with the activation of the unit's net

$$\delta_k = -\frac{\partial J}{\partial net_k} = -\frac{\partial J}{\partial z_k} \cdot \frac{\partial z_k}{\partial net_k} = (t_k - z_k) f'(net_k)$$

Since $net_k = w_k^t.y$ therefore:

$$\frac{\partial net_k}{\partial w_{kj}} = y_j$$

Conclusion: the weight update (or learning rule) for the hidden-to-output weights is:

$$\Delta w_{kj} = \eta \delta_k y_j = \eta (t_k - z_k) f'(net_k) y_j$$

- Error on the input-to-hidden units

$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial y_j} \cdot \frac{\partial y_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ji}}$$

- However, $$\frac{\partial J}{\partial y_j} = \frac{\partial}{\partial y_j}\left[\frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2\right] = -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial y_j}$$
$$= -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial net_k}\cdot\frac{\partial net_k}{\partial y_j} = -\sum_{k=1}^{c}(t_k - z_k)f'(net_k)w_{kj}$$

- Similarly as in the preceding case, we define the sensitivity for a hidden unit:
$$d_j \equiv f'(net_j)\sum_{k=1}^{c} w_{kj}d_k$$

- which means that:"The sensitivity at a hidden unit is simply the sum of the individual sensitivities at the output units weighted by the hidden-to-output weights w_{kj}; all multipled by $f'(net_j)$"

- Conclusion: The learning rule for the input-to-hidden weights is:
$$\Delta w_{ji} = b\,x_i d_j = b\underbrace{\left[\sum w_{kj}d_k\right]}_{d_j} f'(net_j)x_i$$

- Starting with a pseudo-random weight configuration, the stochastic backpropagation algorithm can be written as:

```
Begin  initialize    n_H; w, criterion θ, η,
                     m ← 0
       do m ← m + 1
            x^m ← randomly chosen pattern
            w_ji ←  w_ji + bδ_j x_i; w_kj ← w_kj + bδ_k y_j
       until ||∇J(w)|| < θ
        return w
End
```

- Stopping criterion
  - The algorithm terminates when the change in the criterion function J(w) is smaller than some preset value θ
  - There are other stopping criteria that lead to better performance than this one
  - So far, we have considered the error on a single pattern, but we want to consider an error defined over the entire of patterns in the training set
  - The total training error is the sum over the errors of n individual patterns

$$J = \sum_{p=1}^{n} J_p \qquad (1)$$

- Stopping criterion (cont.)
  - A weight update may reduce the error on the single pattern being presented but can increase the error on the full training set
  - However, given a large number of such individual updates, the total error of equation (1) decreases

- Learning Curves
  - Before training starts, the error on the training set is high; through the learning process, the error becomes smaller
  - The error per pattern depends on the amount of training data and the expressive power (such as the number of weights) in the network
  - The average error on an independent test set is always higher than on the training set, and it can decrease as well as increase
  - A validation set is used in order to decide when to stop training ; we do not want to overfit the network and decrease the power of the classifier generalization

    “we stop training at a minimum of the error on the validation set”

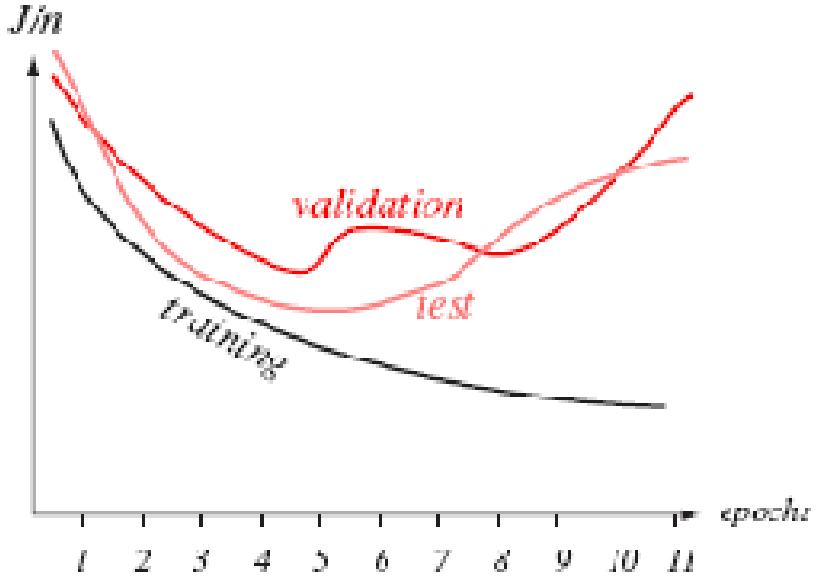


FIGURE 6.6. A learning curve shows the criterion function as a function of the amount of training, typically indicated by the number of epochs or presentations of the full training set. We plot the average error per pattern, that is, $1/n \sum_{p=1}^{n} J_p$. The validation error and the test or generalization error per pattern are virtually always higher than the training error. In some protocols, training is stopped at the first minimum of the validation set. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.