Multi layer perceptron

Introduction

- The LMS algorithm, in particular, provided a powerful gradient descent method for reducing the error, even when the patterns are not linearly separable.

- With a clever choice of nonlinear ϕ *functions, however, we can obtain arbitrary* decisions, in particular the one leading to minimum error.

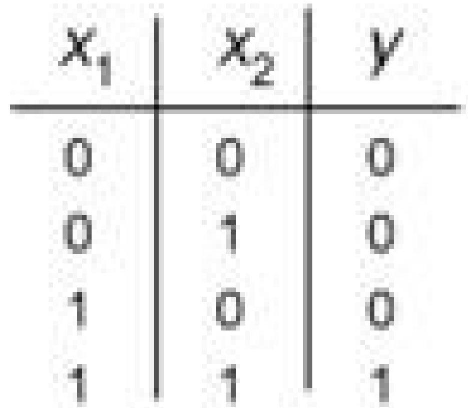
- One brute force approach might be to choose a complete basis set (all polynomials, say) but this will not work; such a classifier would have too many free parameters to be determined from a limited number of training patterns

- Alternatively, we may have prior knowledge relevant to the classification problem and this might guide our choice of nonlinearity.

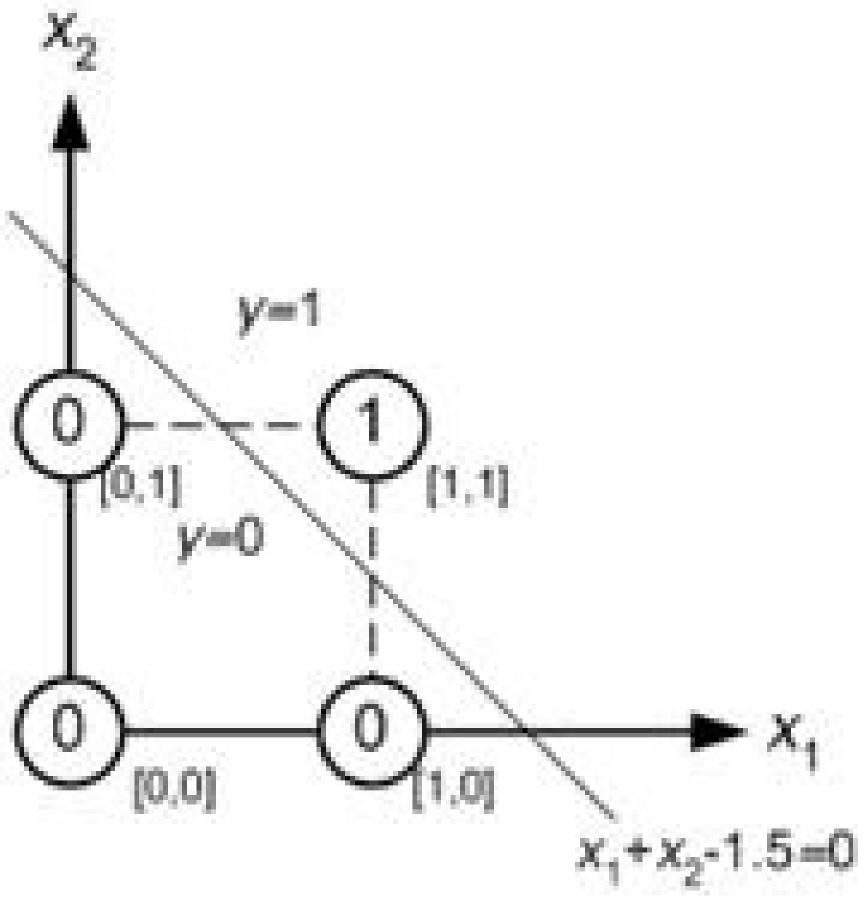
- In the absence of such information, up to now we have seen no principled or automatic method for finding the nonlinearities. What we seek, then, is a way to *learn* the nonlinearity at the same time as the linear discriminant. This is the approach of multilayer neural networks (also called multilayer Perceptrons):

- The parameters governing the nonlinear mapping are learned at the same time as those governing the linear discriminant.

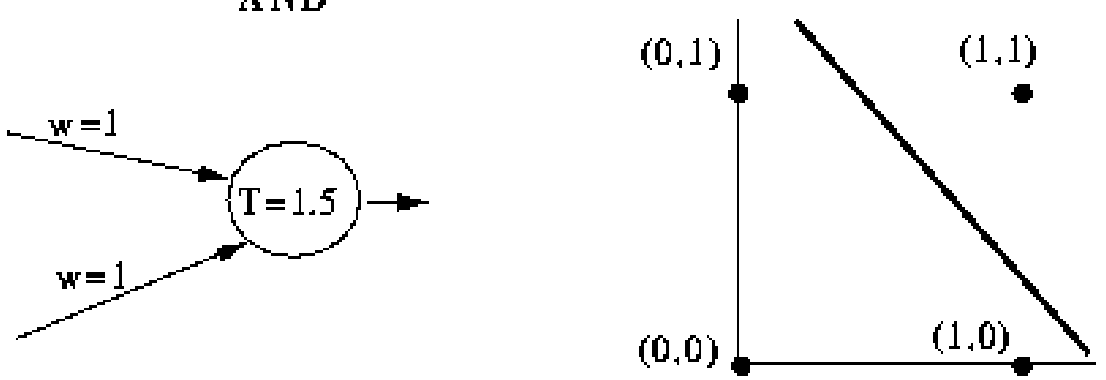
OR gate



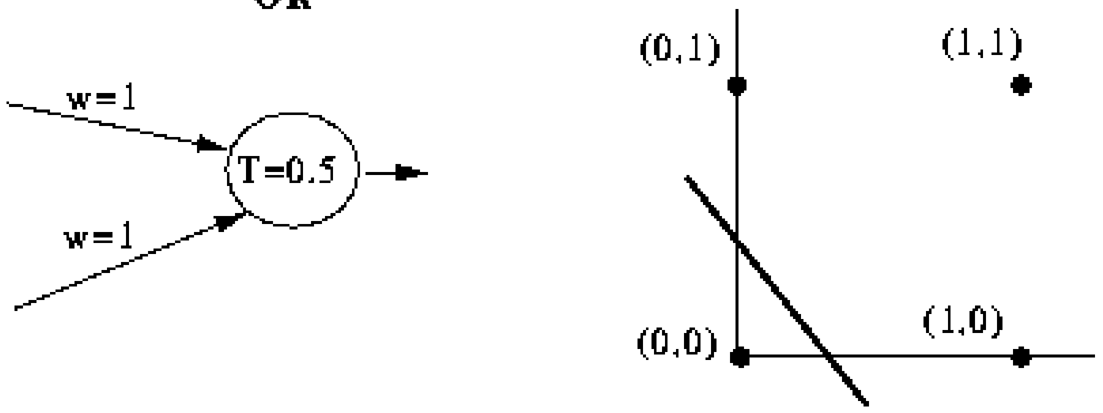
x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1



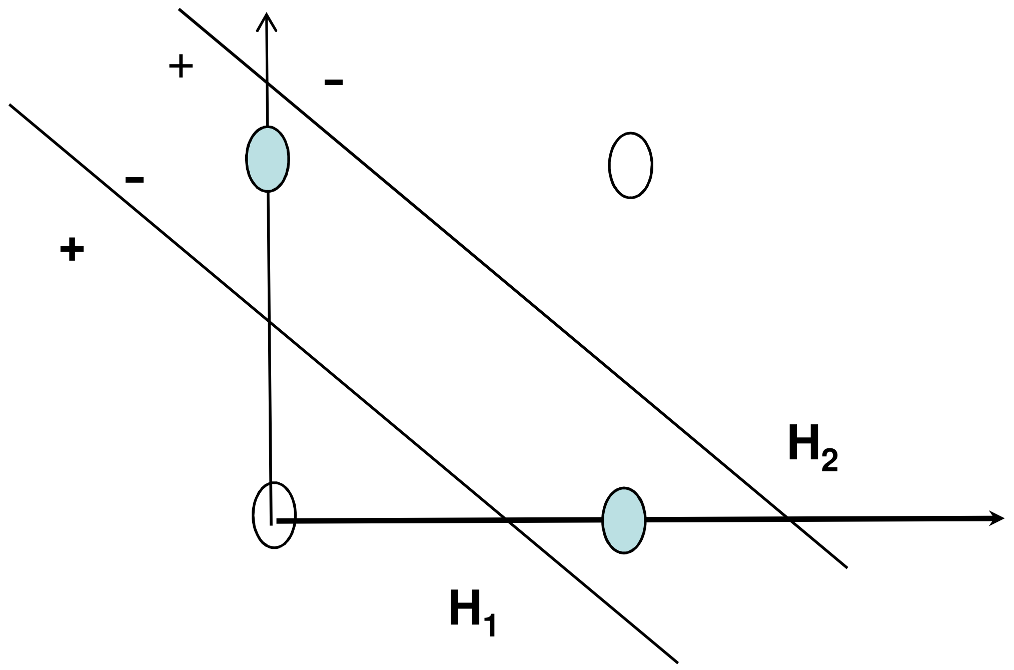
AND



OR

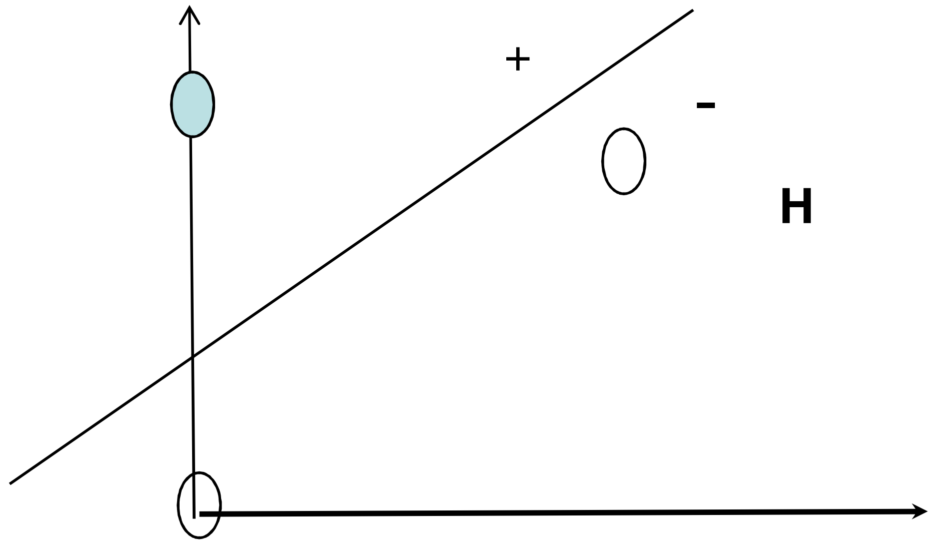


Need of multi layer perceptron : XOR Gate



			H_1	H_2
0	0	0	1	1
0	1	1	0	1
1 1	0 1	1 0	0 0	1 0

Need of multi layer perceptron : XOR Gate



H1	H2	H
1	1	0
0	1	1
0 0	1 0	1 0

- One of the most popular methods for training such multilayer networks is based on gradient descent in error — the back propagation algorithm (or generalized delta rule), a natural extension of the LMS algorithm.

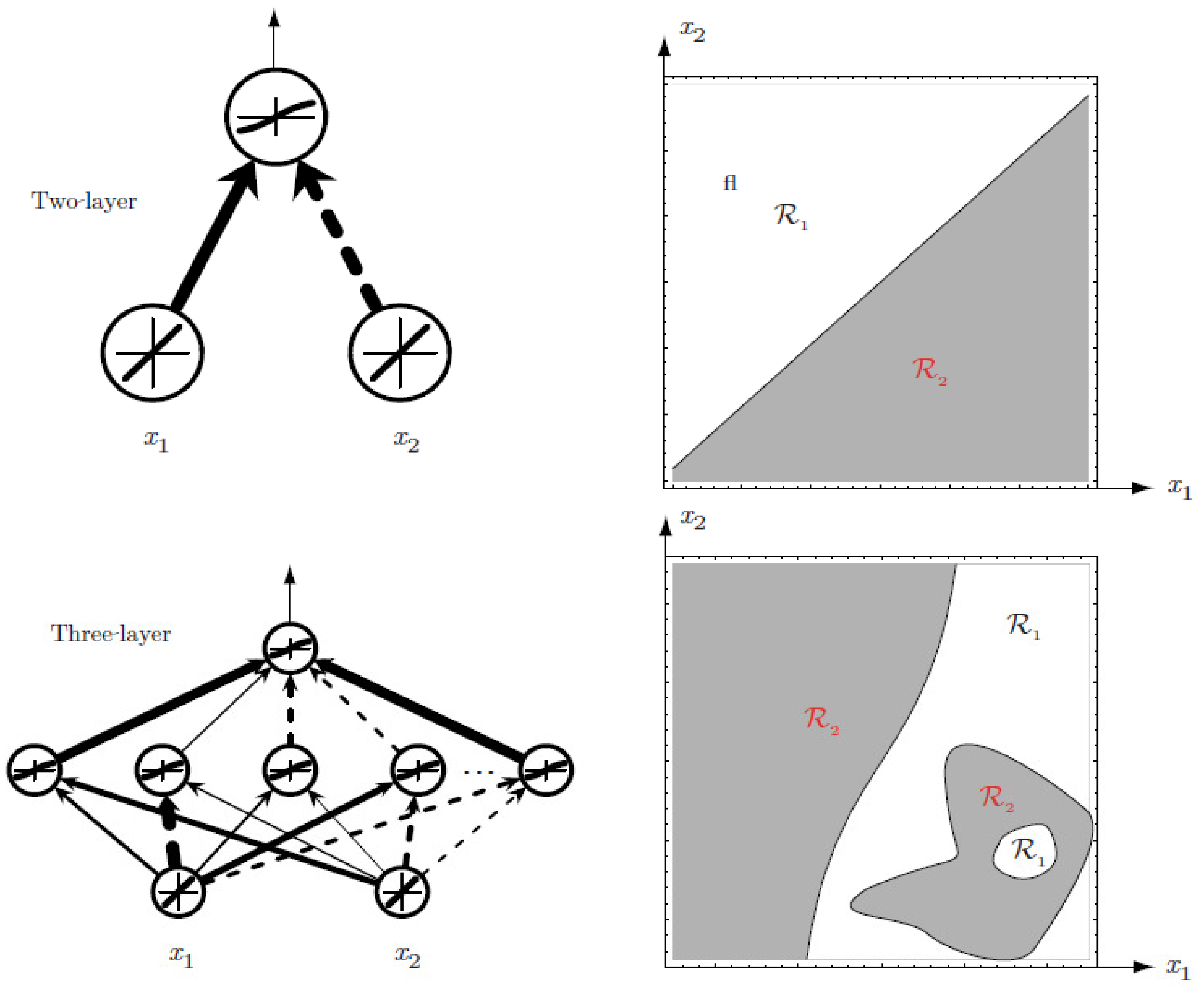
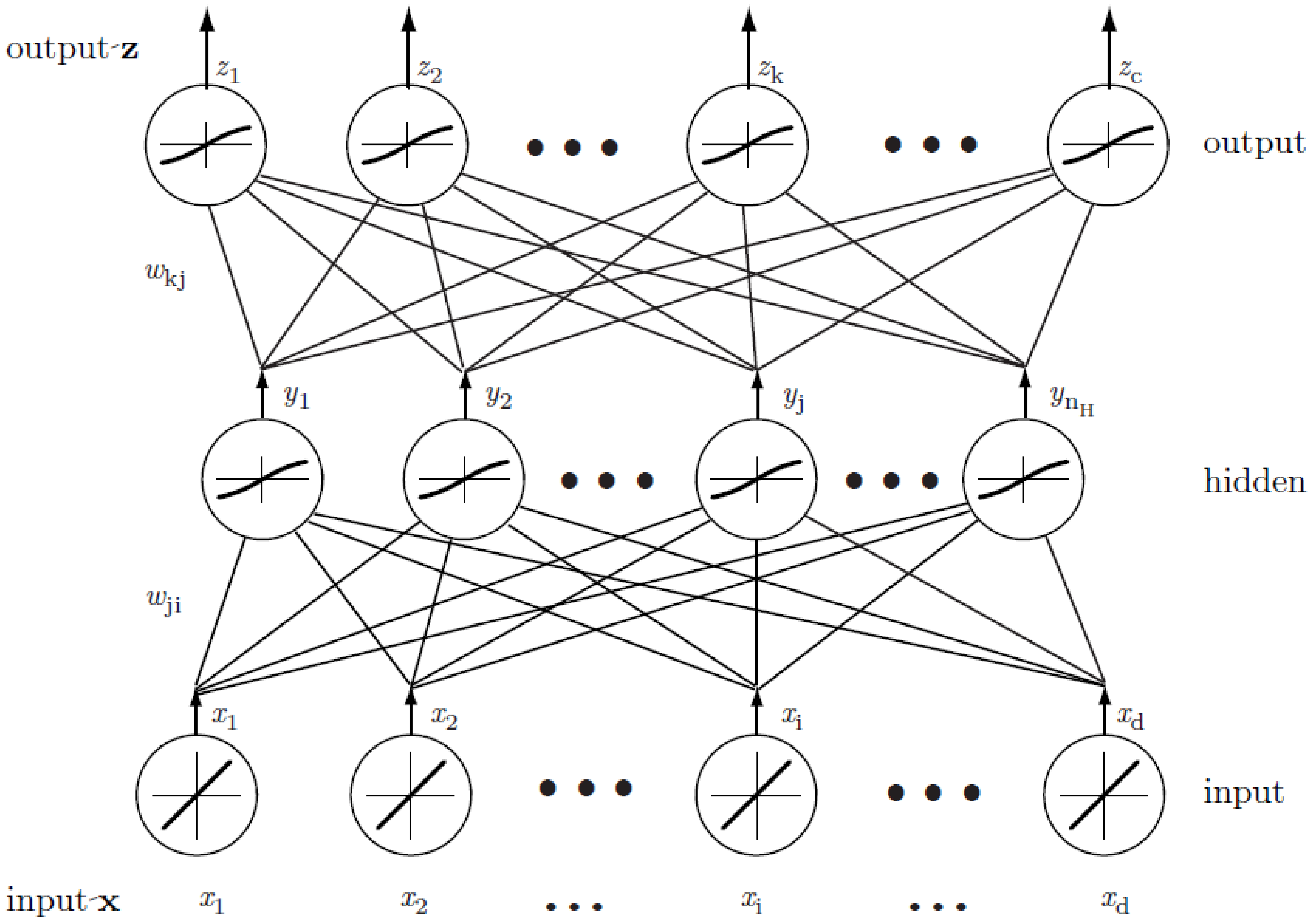


Figure 6.3: Whereas a two-layer network classifier can only implement a linear decision boundary, given an adequate number of hidden units, three-, four- and higher-layer networks can implement arbitrary decision boundaries. The decision regions need not be convex, nor simply connected.

Feedforward operation and classification

- Simple three-layer neural network.

- This one consists of an input layer (having *d* input units), a hidden layer with (n_H hidden units) and an output layer (a single unit), interconnected by modifiable weights, represented by links between layers.


output z
z_1
z_2
z_k
z_c
output
w_{kj}
y_1
y_2
y_j
y_{n_H}
hidden
w_{ji}
x_1
x_2
x_i
x_d
input
input x
x_1
x_2
x_i
x_d

Multi layer perceptron

Net Output at input to *j* th hidden node

$$net_j = \sum_{i=1}^{d} x_i w_{ji} + w_{j0} = \sum_{i=0}^{d} x_i w_{ji} \equiv \mathbf{w}_j^t \mathbf{x},$$

Non linearity imposed by activation function *f*(…)

$$y_j = f(net_j).$$

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

This $f()$ is sometimes called the *transfer function* or merely "nonlinearity" of a unit,

Multi layer perceptron

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

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

where the subscript k indexes units in the output layer

n_H denotes the number of hidden units

Multi layer perceptron

Each output unit then computes the nonlinear function of its *net*, emitting

$$z_k = f(net_k).$$

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

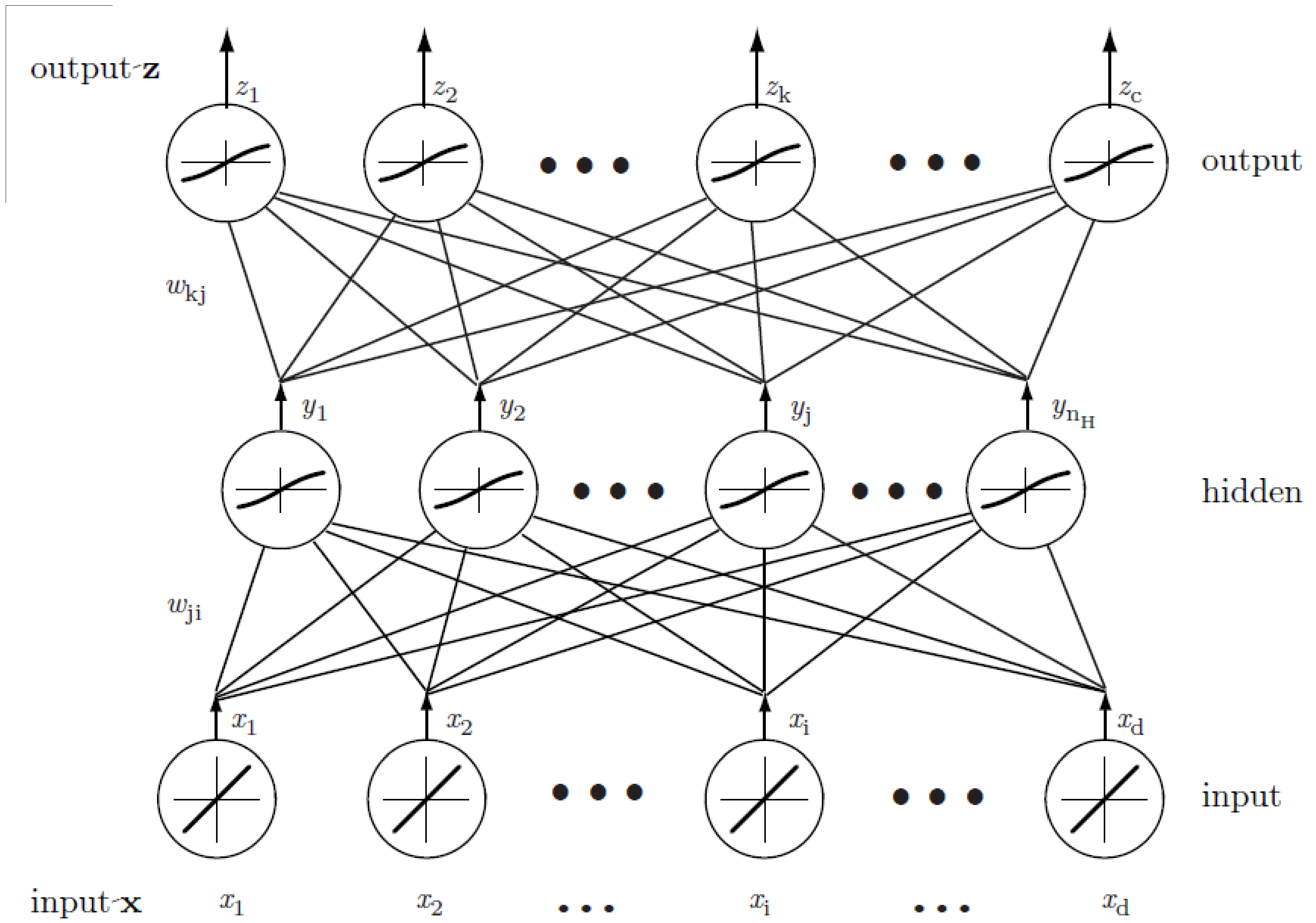
Multi layer perceptron

$$J(\mathbf{w}) \equiv 1/2 \sum_{k=1}^{c} (t_k - z_k)^2 = 1/2(\mathbf{t} - \mathbf{z})^2,$$

where **t** and **z** are the target and the network output vectors of length c; **w** represents all the weights in the network.

The backpropagation learning rule is based on gradient descent. The weights are initialized with random values, and are changed in a direction that will reduce the error:

$$\Delta \mathbf{w} = -\eta \frac{\partial J}{\partial \mathbf{w}},$$



output $\mathbf{z}$
z_1
z_2
z_k
z_c
output
w_{kj}
y_1
y_2
y_j
y_{n_H}
hidden
w_{ji}
x_1
x_2
x_i
x_d
input
input $\mathbf{x}$
x_1
x_2
x_i
x_d

Multi layer perceptron

$$\Delta w_{mn} = -\eta \frac{\partial J}{\partial w_{mn}},$$

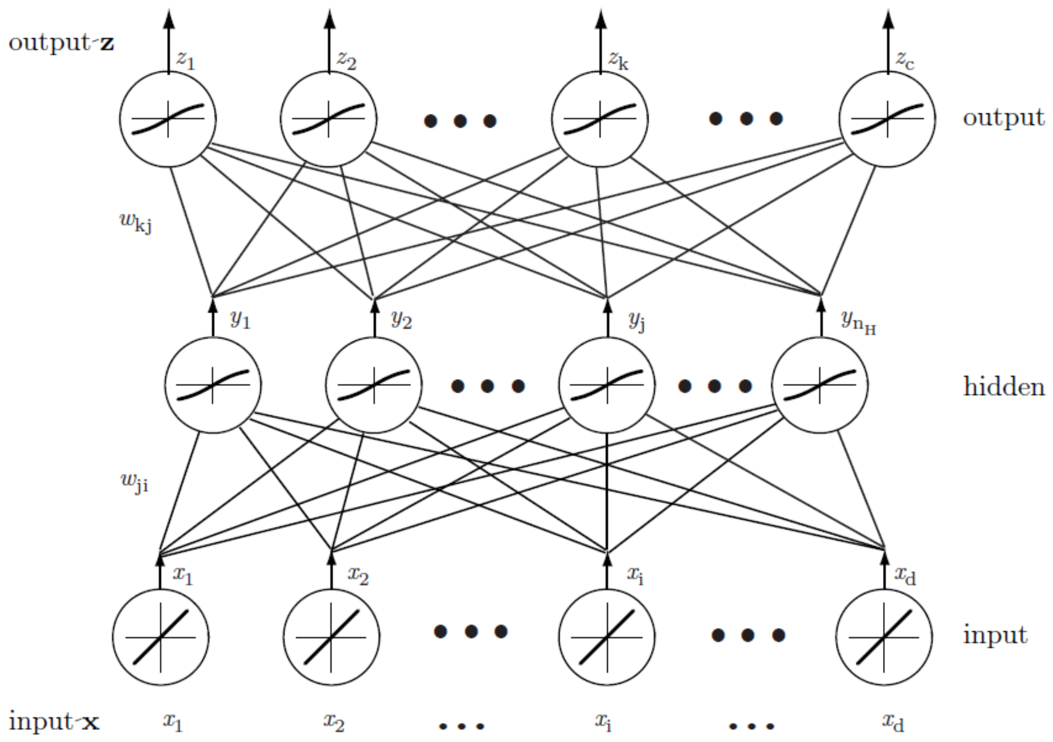
where η is the *learning rate*, and merely indicates the relative size of the change in weights.

Multi layer perceptron

This iterative algorithm requires taking a weight vector at iteration m and updating it as:

$$\mathbf{w}(m+1) = \mathbf{w}(m) + \Delta\mathbf{w}(m),$$

Gradient descent : Back propagation



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

where the *sensitivity* of unit k is defined to be

$$\delta_k \equiv -\partial J/\partial net_k,$$

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

$$\delta_k \equiv -\frac{\partial J}{\partial net} = -\frac{\partial J}{\partial z}\frac{\partial z_k}{\partial net} = (t_k - z_k)f'(net_k).$$

Gradient descent : Back propagation

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

Taken together, these results give the weight update (learning rule) for the hidden-to-output weights:

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

Gradient descent : Back propagation

The learning rule for the input-to-hidden units is more subtle,

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

$$\begin{aligned}
\frac{\partial J}{\partial y_j} &= \frac{\partial}{\partial y_j}\left[1/2\sum_{k=1}^{c}(t_k - z_k)^2\right] \\
&= -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial y_j} \\
&= -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial net_k}\frac{\partial net_k}{\partial y_j} \\
&= -\sum_{k=1}^{c}(t_k - z_k)f'(net_k)w_{jk}.
\end{aligned}$$

Gradient descent : Back propagation

$$\delta_j \equiv f'(net_j) \sum_{k=1}^{c} w_{kj}\delta_k.$$

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_{jk}, all multiplied by $f'(net_j)$. Thus the learning rule for the input-to-hidden weights is:

$$\Delta w_{ji} = \eta x_i \delta_j = \eta x_i f'(net_j) \sum_{k=1}^{c} w_{kj}\delta_k.$$

Gradient descent : Back propagation

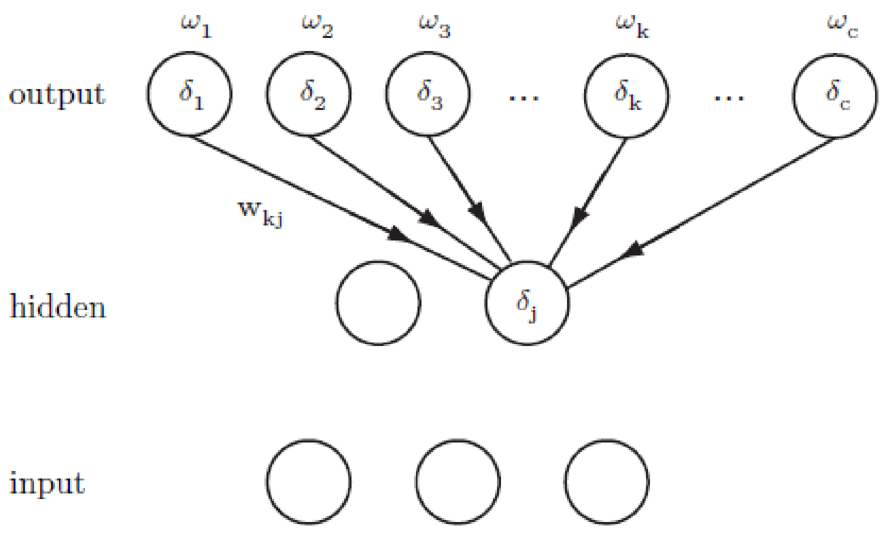
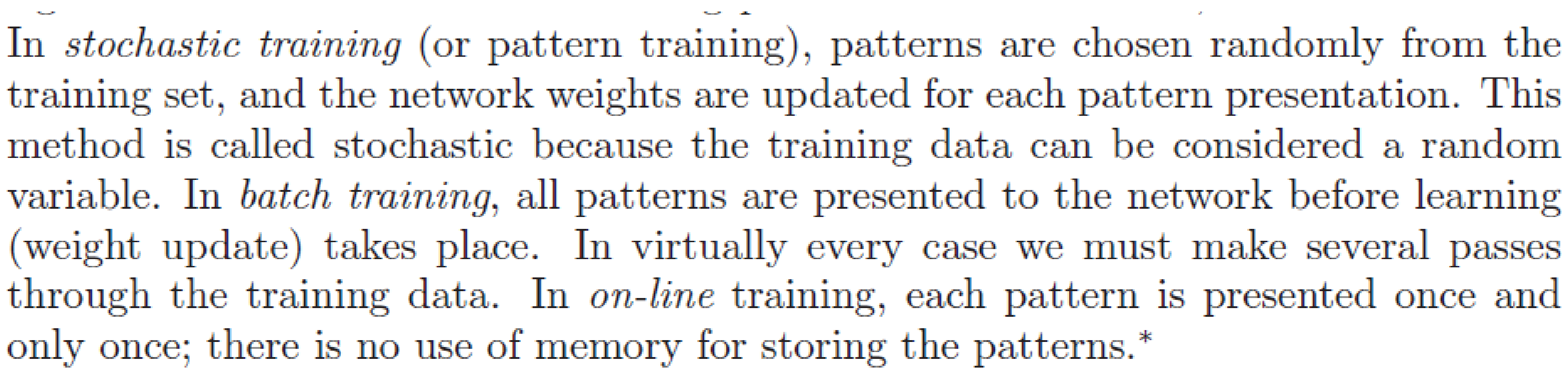
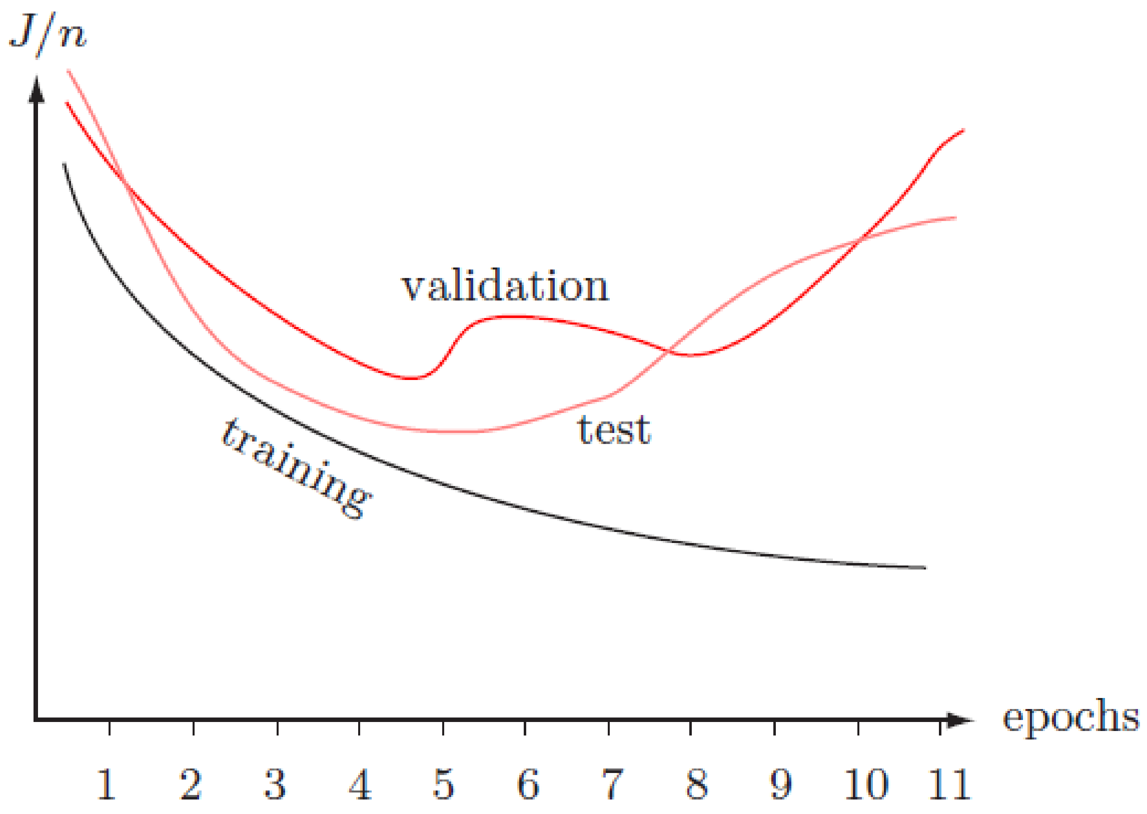


Figure 6.5: The sensitivity at a hidden unit is proportional to the weighted sum of the sensitivities at the output units: $\delta_j = f'(net_j) \sum_{k=1}^{c} w_{kj}\delta_k$. The output unit sensitivities are thus propagated "back" to the hidden units.

Training methodologies



In *stochastic training* (or pattern training), patterns are chosen randomly from the training set, and the network weights are updated for each pattern presentation. This method is called stochastic because the training data can be considered a random variable. In *batch training*, all patterns are presented to the network before learning (weight update) takes place. In virtually every case we must make several passes through the training data. In *on-line* training, each pattern is presented once and only once; there is no use of memory for storing the patterns.*


J/n
validation
test
training
epochs
1
2
3
4
5
6
7
8
9
10
11

Activation function

One class of function that has all the above properties is the *sigmoid* such as a hyperbolic tangent. The sigmoid is smooth, differentiable, nonlinear, and saturating.

A hidden layer of sigmoidal units affords a *distributed* or *global* representation of the input. That is, any particular input $\mathbf{x}$ is likely to yield activity throughout *several* hidden units. In contrast, if the hidden units have transfer functions that have significant response only for inputs within a small range, then an input $\mathbf{x}$ generally leads to *fewer* hidden units being active — a *local representation.*

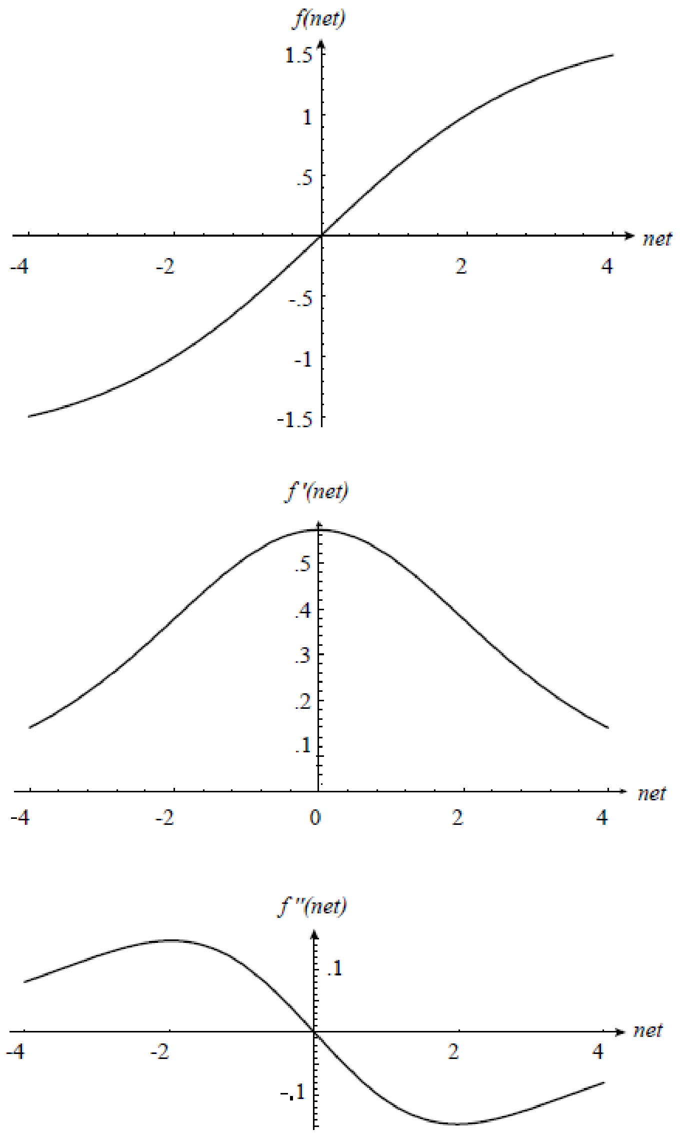


Figure 6.16: A useful transfer function $f(net)$ is an anti-symmetric sigmoid. For the parameters given in the text, $f(net)$ is nearly linear in the range $-1 < net < +1$ and its second derivative, $f''(net)$, has extrema near $net \simeq \pm 2$.