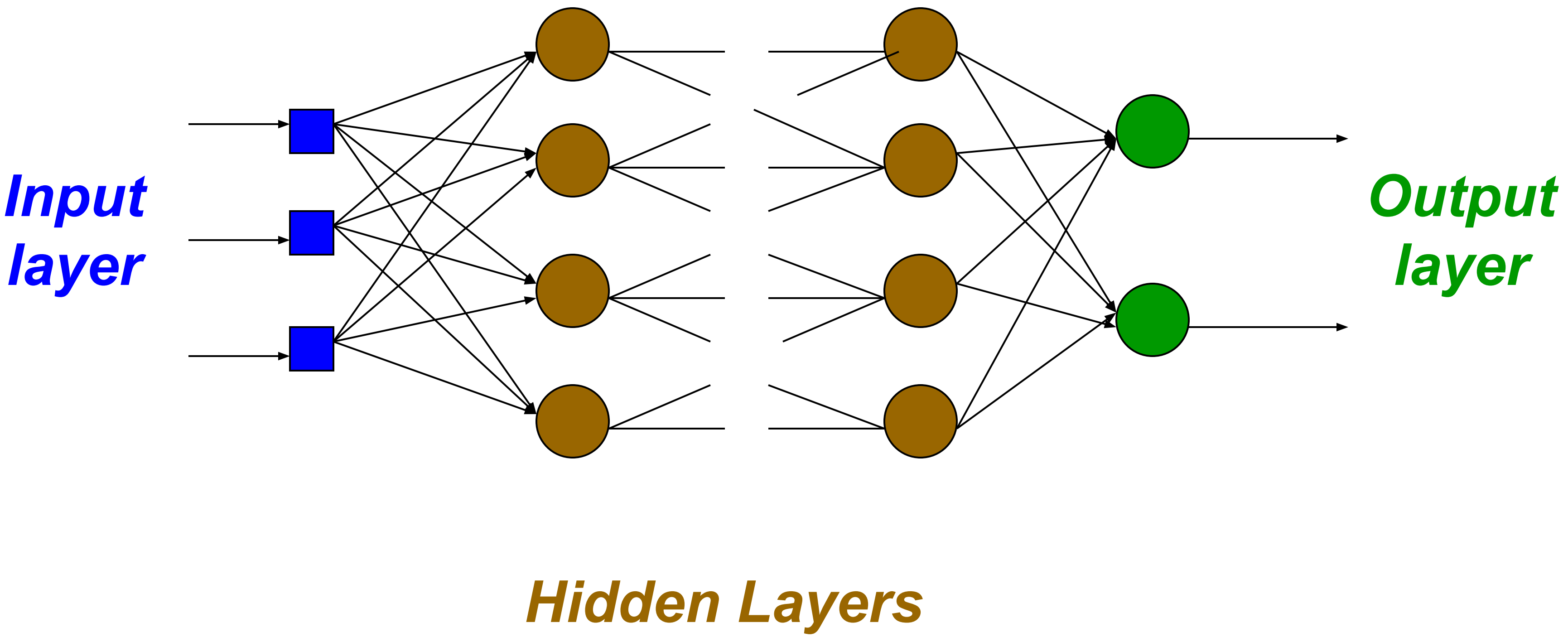
# Multilayer Percetrons

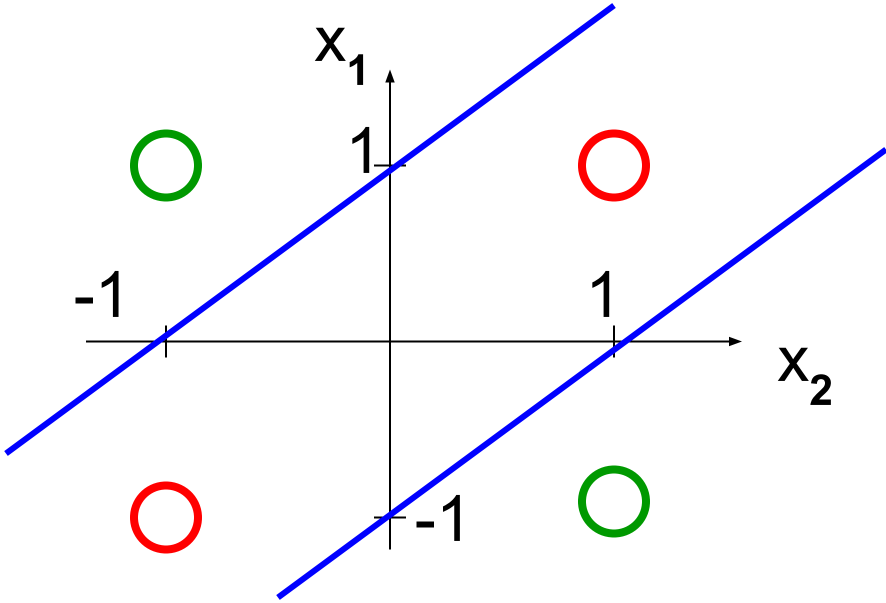
Neural Networks, Simon Haykin, Prentice-Hall, 3rd edition

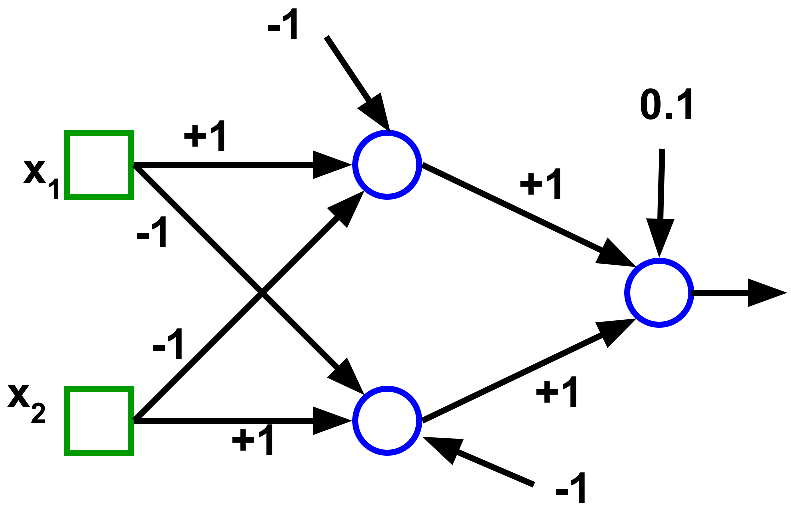
# Multilayer Perceptrons Architecture



# A solution for the XOR problem

| $x_1$ | $x_2$ | $x_1$ xor $x_2$ |
|---|---|---|
| -1 | -1 | **-1** |
| -1 | 1 | **1** |
| 1 | -1 | **1** |
| 1 | 1 | **-1** |



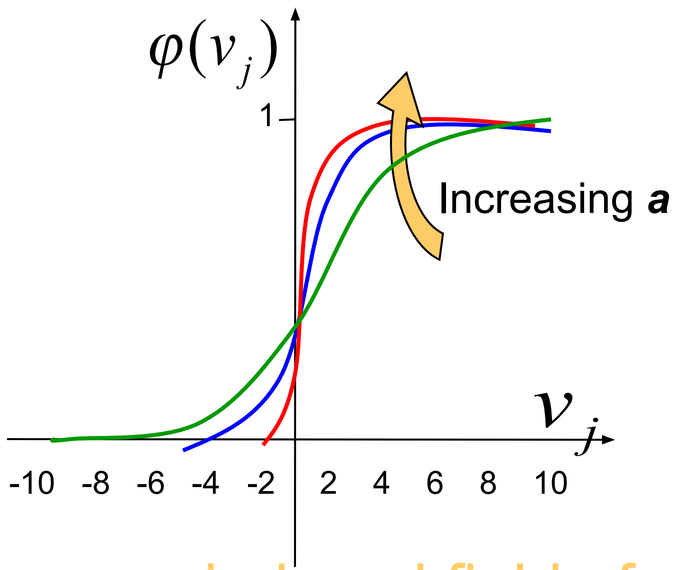




$$\phi(v) = \begin{cases} 1 & \text{if } v > 0 \\ -1 & \text{if } v \leq 0 \end{cases}$$

$\phi$ is the sign function.

# NEURON MODEL

- Sigmoidal Function



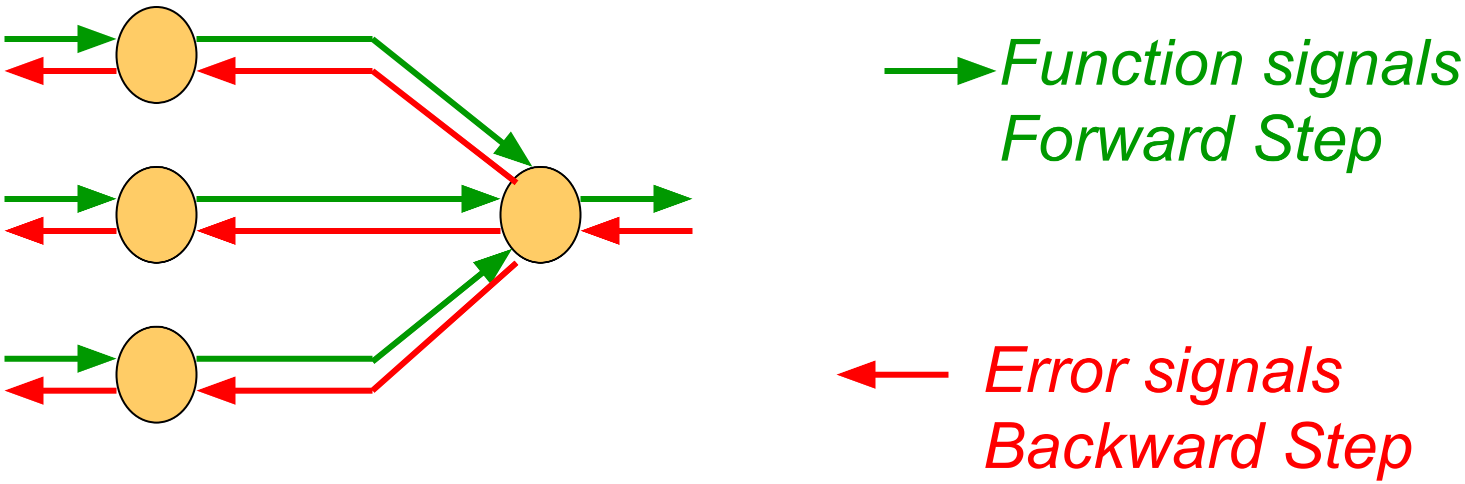

$$\boxed{\varphi(\mathrm{v}_j) = \frac{1}{1+e^{-a v_j}}}$$

$$\mathrm{v}_j = \sum_{i=0,\dots,m} w_{ji} y_i$$

- $\mathrm{v}_j$ induced field of neuron j
- Most common form of activation function
- a $\rightarrow \infty \Rightarrow \phi \rightarrow$ threshold function
- Differentiable

# LEARNING ALGORITHM

- Back-propagation algorithm




- It adjusts the weights of the NN in order to minimize the average squared error.

# Average Squared Error

- Error signal of output neuron ***j*** at presentation of ***n-th*** training example:

$$\boxed{e_j(n) = d_j(n) - y_j(n)}$$

- Total energy at time ***n***:

$$\boxed{E(n) = \frac{1}{2}\sum_{j\in C} e_j^2(n)}$$

- Average squared error:
- Measure of learning performance:

$$\boxed{E_{AV} = \frac{1}{N}\sum_{n=1}^{N} E(n)}$$

*C: Set of neurons in output layer*

*N: size of training set*

- **Goal:** ***Adjust weights of NN to minimize $E_{AV}$***

# Notation

$e_j$ Error at output of neuron j

$y_j$ Output of neuron j

$$v_j = \sum_{i=0,\ldots,m} w_{ji} y_i$$ Induced local field of neuron j

# Weight Update Rule

Update rule is based on the gradient descent method take a step in the direction yielding the maximum decrease of E

$$\boxed{\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}}}$$

*Step in direction opposite to the gradient*

With $w_{ji}$ weight associated to the link from neuron i to neuron j

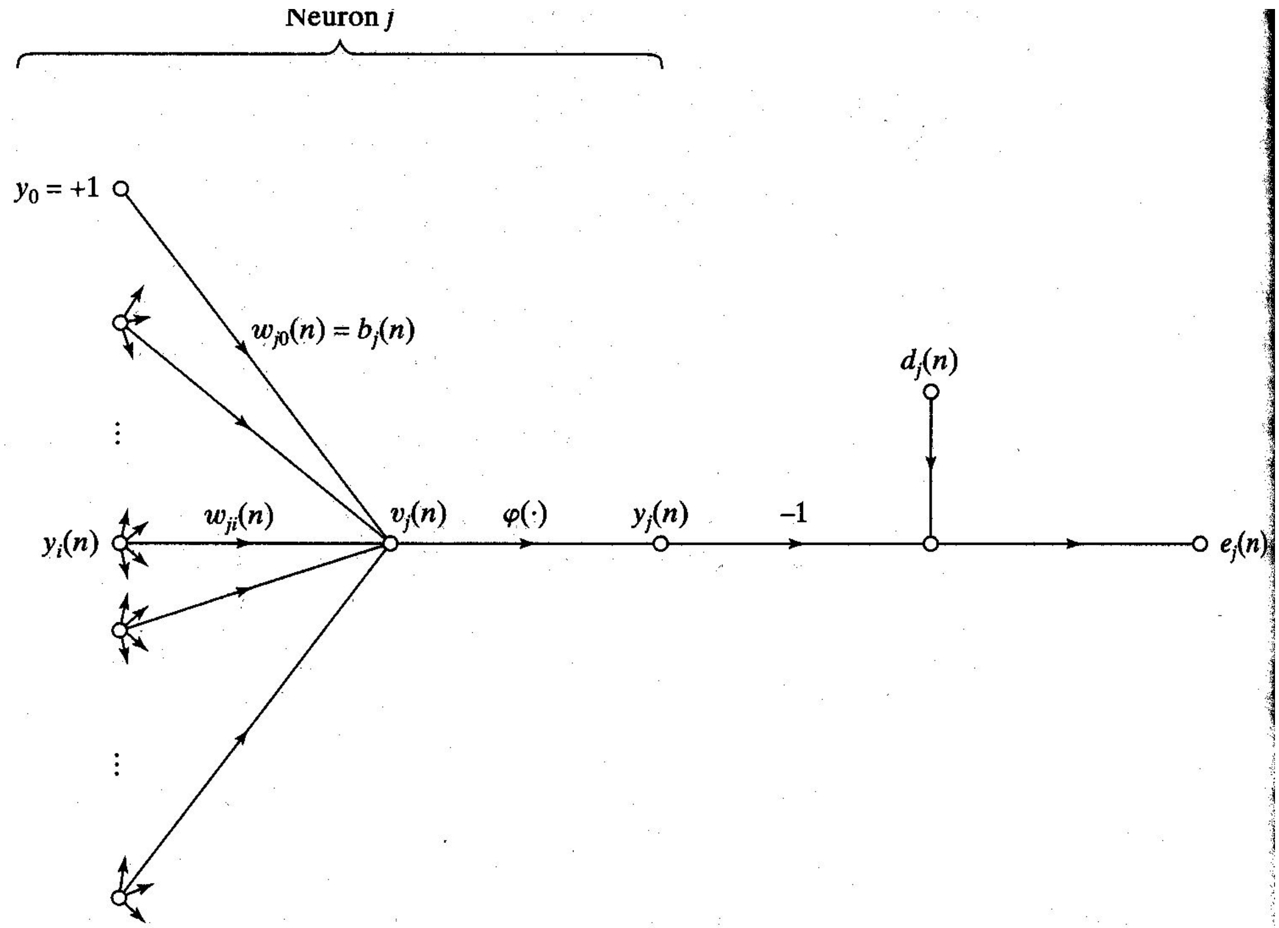



**FIGURE 4.3** Signal-flow graph highlighting the details of output neuron $j$.

# Definition of the Local Gradient of neuron j

$$\boxed{\delta_j = -\frac{\partial E}{\partial v_j}}$$

**Local Gradient**

We obtain $\boxed{\delta_j = e_j \varphi'(v_j)}$

because

$$-\frac{\partial E}{\partial v_j} = -\frac{\partial E}{\partial e_j}\frac{\partial e_j}{\partial y_j}\frac{\partial y_j}{\partial v_j} = -e_j(-1)\varphi'(v_j)$$

# Update Rule

- We obtain $$\boxed{\Delta \mathrm{w}_{\mathrm{ji}} = \eta \delta_j y_i}$$

because

$$\frac{\partial E}{\partial \mathrm{w}_{\mathrm{ji}}} = \frac{\partial E}{\partial \mathrm{v}_{\mathrm{j}}} \frac{\partial \mathrm{v}_{\mathrm{j}}}{\partial \mathrm{w}_{\mathrm{ji}}}$$

$$-\frac{\partial E}{\partial \mathrm{v}_j} = \delta_j \qquad \frac{\partial v_j}{\partial \mathrm{w}_{\mathrm{ji}}} = y_i$$

# Compute local gradient of neuron j

- The key factor is the calculation of $\boldsymbol{e_j}$
- There are two cases:
  - Case 1): $\boldsymbol{j}$ is a output neuron
  - Case 2): $\boldsymbol{j}$ is a hidden neuron

# Error $e_j$ of output neuron

- Case 1: ***j output neuron***

$$e_j = d_j - y_j$$

Then

$$\boxed{\delta_j = (d_j - y_j)\varphi'(v_j)}$$

# Local gradient of hidden neuron

- Case 2: ***j hidden neuron***

- the local gradient for neuron j is recursively determined in terms of the local gradients of all neurons to which neuron j is directly connected

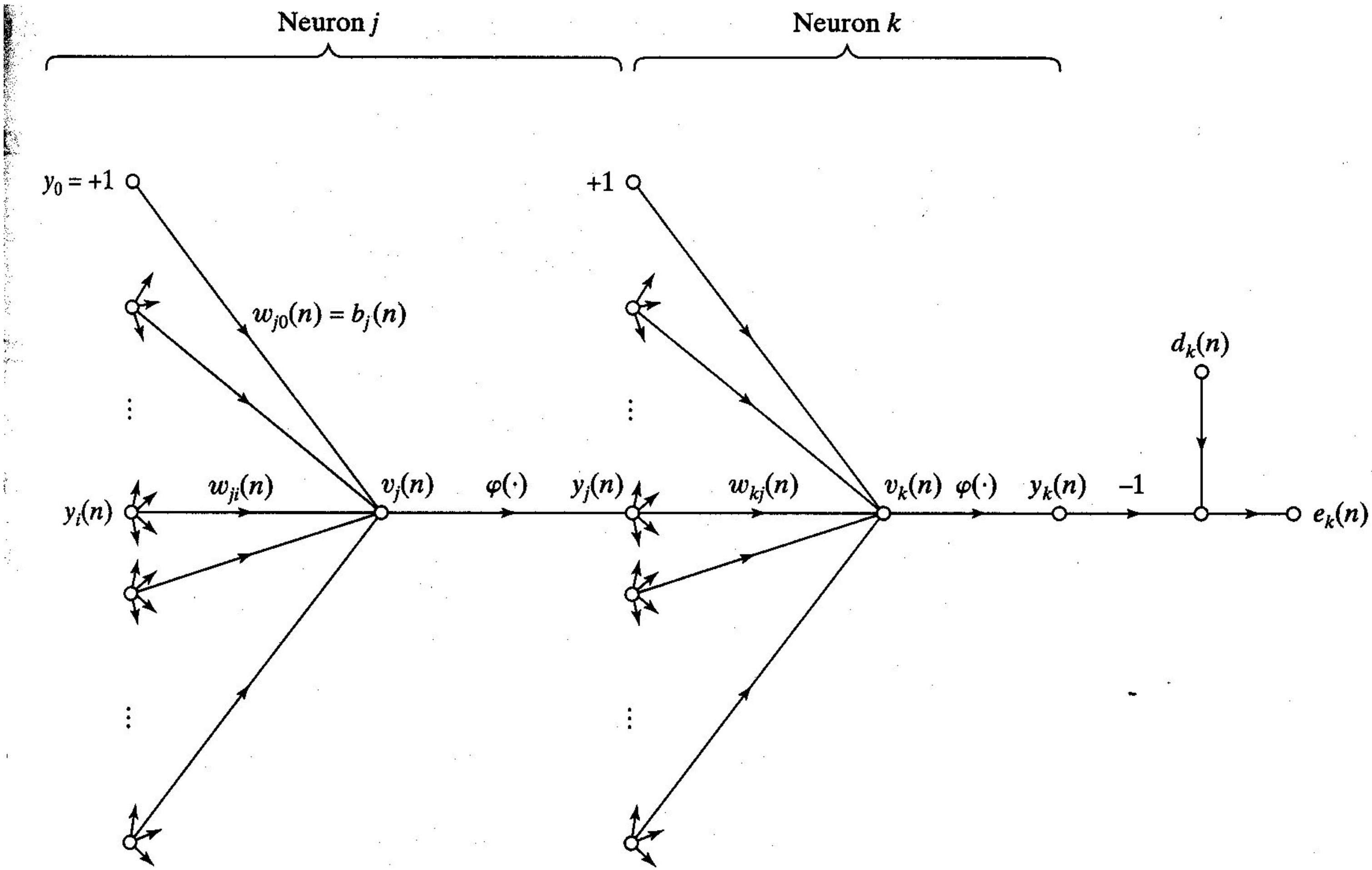



**FIGURE 4.4** Signal-flow graph highlighting the details of output neuron $k$ connected to hidden neuron $j$.

# Use the Chain Rule

$$\delta_j = -\frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial v_j} \qquad \frac{\partial y_j}{\partial v_j} = \varphi'(v_j)$$

$$\boxed{E(n) = \frac{1}{2}\sum_{k \in C} e_k^2(n)}$$

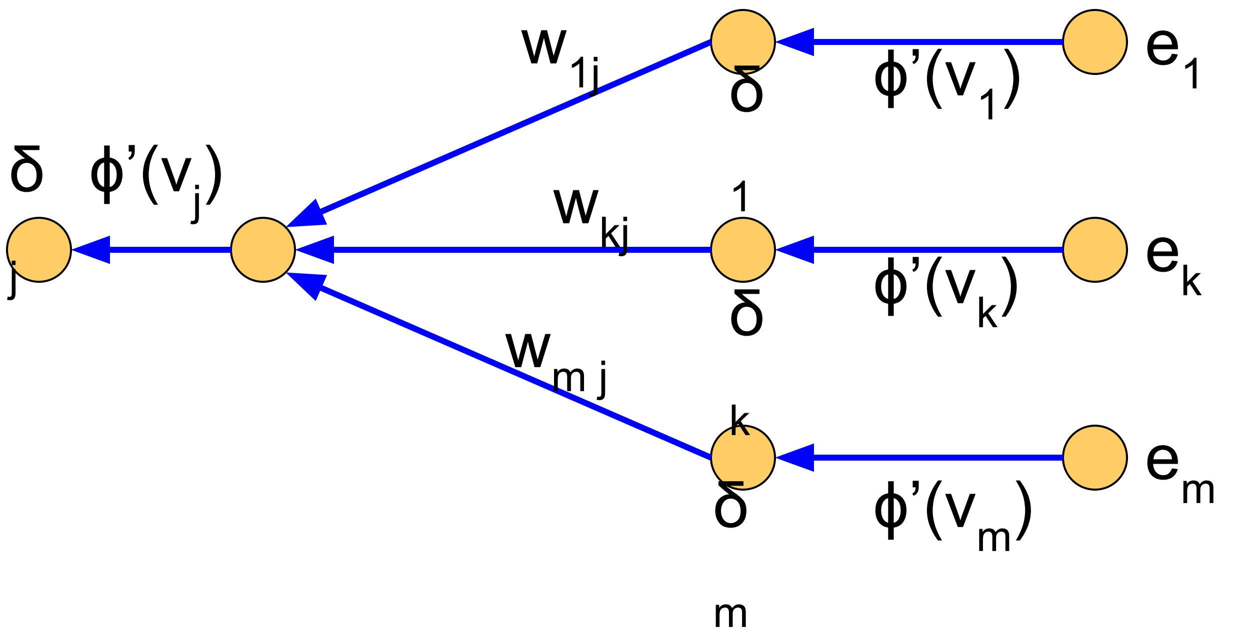
$$-\frac{\partial E}{\partial y_j} = -\sum_{k \in C} e_k \frac{\partial e_k}{\partial y_j} = \sum_{k \in C} e_k \left[\frac{-\partial e_k}{\partial v_k}\right]\frac{\partial v_k}{\partial y_j}$$

from $$-\frac{\partial e_k}{\partial v_k} = \varphi'(v_k) \qquad \frac{\partial v_k}{\partial y_j} = w_{kj}$$

We obtain $$-\frac{\partial E}{\partial y_j} = \sum_{k \in C} \delta_k w_{kj}$$

# Local Gradient of hidden neuron j

Hence
$$\boxed{\delta_j = \varphi'(v_j) \sum_{k \in C} \delta_k w_{kj}}$$




Signal-flow graph of back-propagation error signals to neuron ***j***

# Delta Rule

- **Delta rule** $\Delta w_{ji} = \eta \delta_j y_i$

$$\delta_j = \begin{cases} \varphi'(v_j)(d_j - y_j) & \text{IF j output node} \\ \varphi'(v_j) \sum_{k \in C} \delta_k w_{kj} & \text{IF j hidden node} \end{cases}$$

C: Set of neurons in the layer following the one containing ***j***

# Local Gradient of neurons

$$\varphi'(v_j) = ay_j[1 - y_j]$$

**a > 0**

$$\delta_j = \begin{cases} ay_j[1 - y_j] \sum \delta_k w_{kj} & \textit{if j hidden node} \\ ay_j[1 - y_j][d_j^k - y_j] & \textit{If j output node} \end{cases}$$

# Backpropagation algorithm

- Two phases of computation:
  - **Forward pass**: run the NN and compute the error for each neuron of the output layer.
  - **Backward pass**: start at the output layer, and pass the errors backwards through the network, layer by layer, by recursively computing the local gradient of each neuron.

# Summary



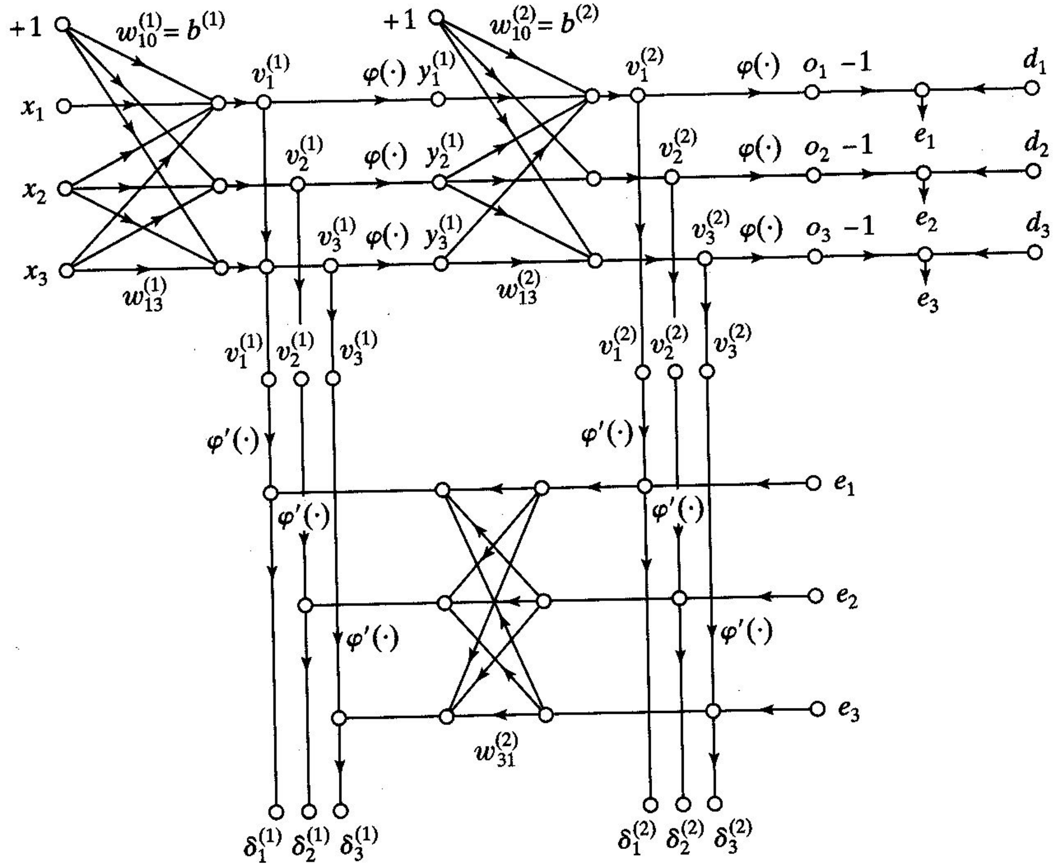



**FIGURE 4.7** Signal-flow graphical summary of back-propagation learning. Top part of the graph: forward pass. Bottom part of the graph: backward pass.

# Training

- **Sequential mode** (on-line, pattern or stochastic mode):
  - (x(1), d(1)) is presented, a sequence of forward and backward computations is performed, and the weights are updated using the delta rule.
  - Same for (x(2), d(2)), … , (x(N), d(N)).

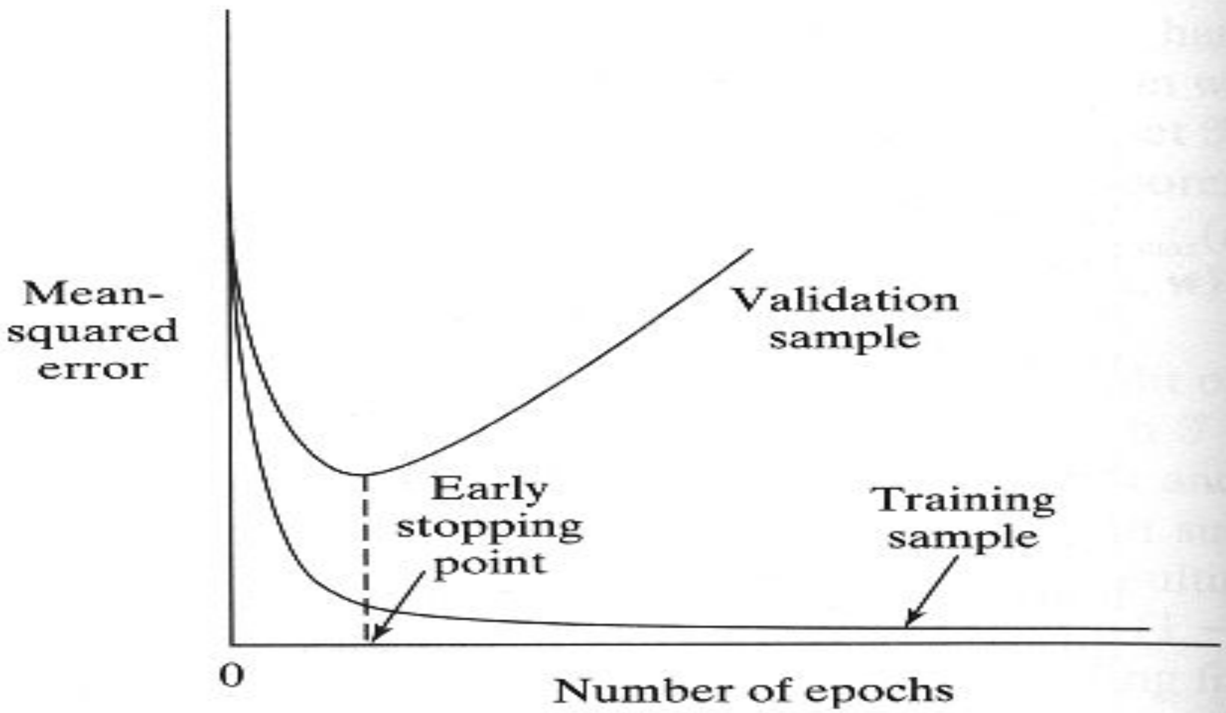
# Training

- The learning process continues on an epoch-by-epoch basis until the stopping condition is satisfied.
- From one epoch to the next choose a randomized ordering for selecting examples in the training set.

# Stopping criterions

- Sensible stopping criterions:
  - Average squared error change: Back-prop is considered to have converged when the absolute rate of change in the average squared error per epoch is sufficiently small (in the range [0.1, 0.01]).
  - Generalization based criterion: After each epoch the NN is tested for generalization. If the generalization performance is adequate then stop.

# Early stopping



# Generalization

- Generalization: NN generalizes well if the I/O mapping computed by the network is nearly correct for new data (test set).
- Factors that influence generalization:
  - the size of the training set.
  - the architecture of the NN.
  - the complexity of the problem at hand.
- Overfitting (overtraining): when the NN learns too many I/O examples it may end up memorizing the training data.

# Generalization

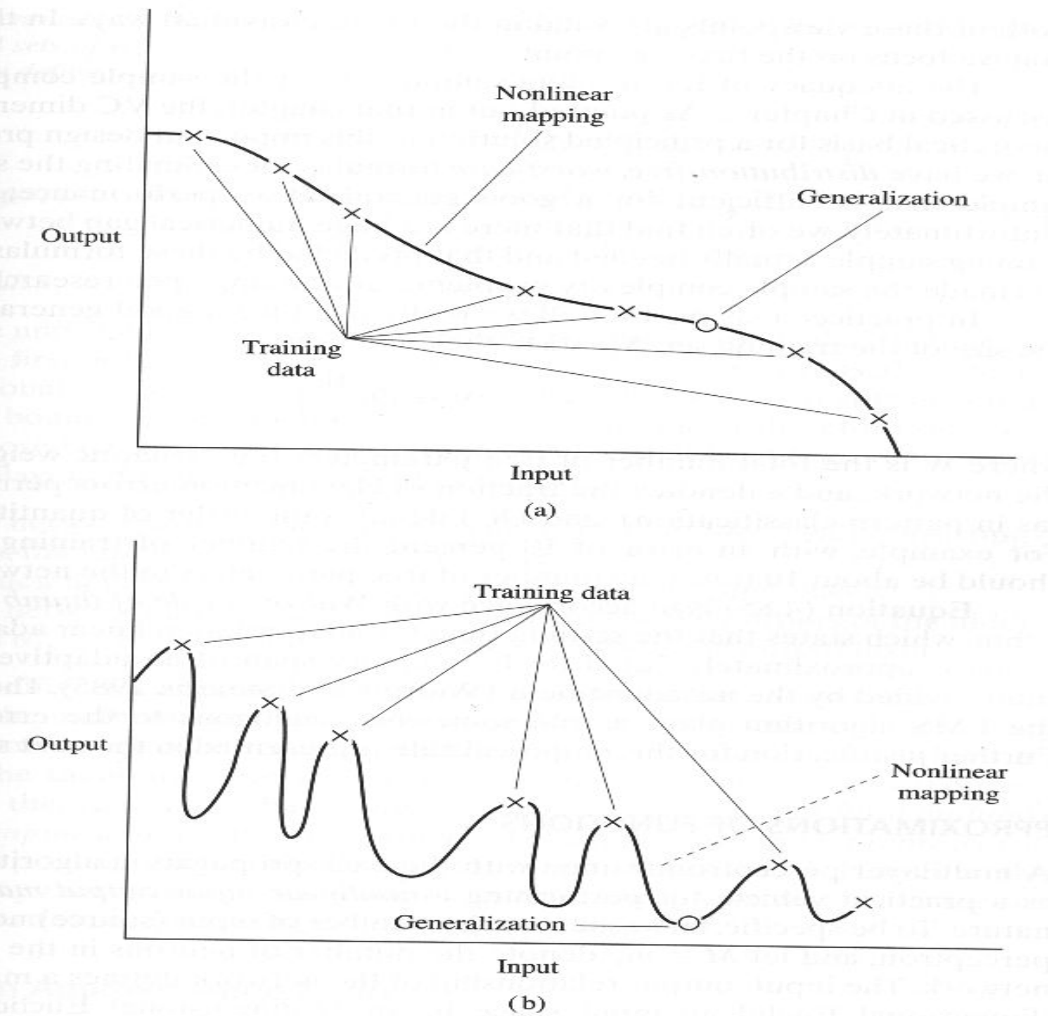



**FIGURE 4.19** (a) Properly fitted data (good generalization) (b) Overfitted data (poor generalization).

# Expressive capabilities of NN

Boolean functions:

- Every boolean function can be represented by network with single hidden layer
- but might require exponential hidden units

Continuous functions:

- Every bounded continuous function can be approximated with arbitrarily small error, by network with one hidden layer
- Any function can be approximated with arbitrary accuracy by a network with two hidden layers

# Generalized Delta Rule

- If η small ⇒ Slow rate of learning

  If η large ⇒ Large changes of weights

  ⇒ NN can become unstable (oscillatory)

- Method to overcome above drawback: **include a momentum term in the delta rule**

$$\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta \delta_j(n) y_i(n)$$

***Generalized delta function***

***momentum constant***

# Generalized delta rule

- the momentum accelerates the descent in steady downhill directions.
- the momentum has a stabilizing effect in directions that oscillate in time.

# $\eta$ adaptation

Heuristics for accelerating the convergence of the back-prop algorithm through $\eta$ adaptation:

- Heuristic 1: Every weight should have its own $\eta$.
- Heuristic 2: Every $\eta$ should be allowed to vary from one iteration to the next.

# NN DESIGN

- Data representation
- Network Topology
- Network Parameters
- Training
- Validation

# Setting the parameters

- How are the weights initialised?
- How is the learning rate chosen?
- How many hidden layers and how many neurons?
- Which activation function ?
- How to preprocess the data ?
- How many examples in the training data set?

# Some heuristics (1)

- Sequential x Batch algorithms: the sequential mode (pattern by pattern) is computationally faster than the batch mode (epoch by epoch)

# Some heuristics (2)

- Maximization of information content: every training example presented to the backpropagation algorithm must maximize the information content.
  - The use of an example that results in the largest training error.
  - The use of an example that is radically different from all those previously used.

# Some heuristics (3)

- Activation function: network learns faster with antisymmetric functions when compared to nonsymmetric functions.

$$\varphi(\mathrm{v}) = \frac{1}{1+e^{-av}}$$

Sigmoidal function is nonsymmetric

$$\varphi(\mathrm{v}) = a\tanh(bv)$$

Hyperbolic tangent function is nonsymmetric

# Some heuristics (3)

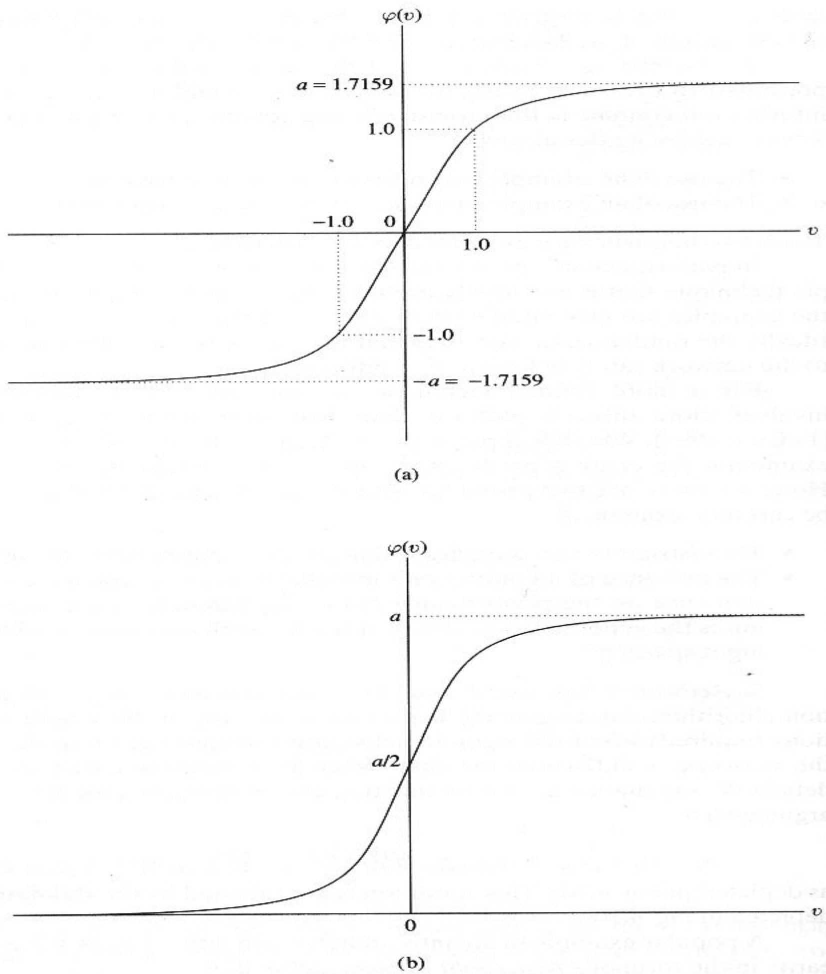



**FIGURE 4.10** Antisymmetric activation function. (b) Nonsymmetric activation function.

# Some heuristics (4)

- Target values: target values must be chosen within the range of the sigmoidal activation function.
- Otherwise, hidden neurons can be driven into saturation which slows down learning

# Some heuristics (4)

- For the antisymmetric activation function it is necessary to design Є
- For a+: $$d_j = a - \varepsilon$$
- For –a: $$d_j = -a + \varepsilon$$
- If a=1.7159 we can set Є=0.7159 then d=±1

# Some heuristics (5)

- Inputs normalisation:
  – Each input variable should be processed so that the mean value is small or close to zero or at least very small when compared to the standard deviation.
  – Input variables should be uncorrelated.
  – Decorrelated input variables should be scaled so their covariances are approximately equal.

# Some heuristics (5)

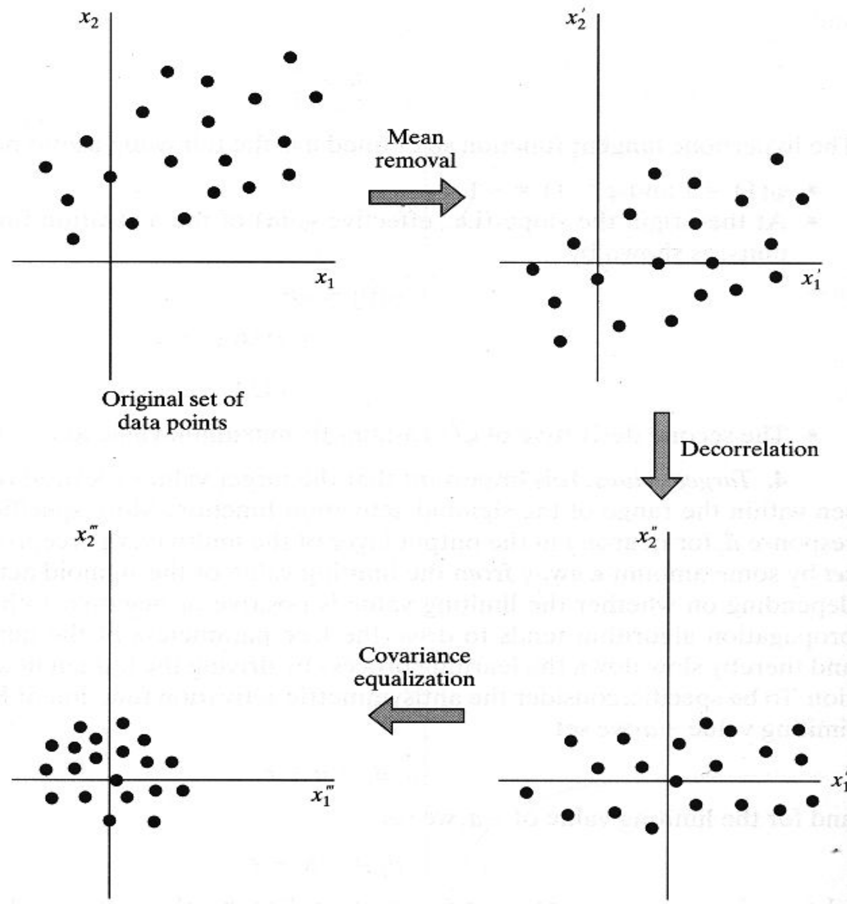



**FIGURE 4.11** Illustrating the operation of mean removal, decorrelation, and covariance equalization for a two-dimensional input space.

# Some heuristics (6)

- Initialisation of weights:
  - If synaptic weights are assigned large initial values neurons are driven into saturation. Local gradients become small so learning rate becomes small.
  - If synaptic weights are assigned small initial values algorithms operate around the origin. For the hyperbolic activation function the origin is a saddle point.

# Some heuristics (6)

- Weights must be initialised for the standard deviation of the local induced field v lies in the transition between the linear and saturated parts.

$$\sigma_v = 1$$

$$\sigma_w = m^{-1/2}$$ m=number of weights

# Some heuristics (7)

- Learning rate:
  - The right value of $\eta$ depends on the application. Values between 0.1 and 0.9 have been used in many applications.
  - Other heuristics adapt $\eta$ during the training as described in previous slides.

# Some heuristics (8)

- How many layers and neurons
  - The number of layers and of neurons depend on the specific task. In practice this issue is solved by trial and error.
  - Two types of adaptive algorithms can be used:
    - **start from a large network and successively remove some neurons and links until network performance degrades.**
    - **begin with a small network and introduce new neurons until performance is satisfactory.**

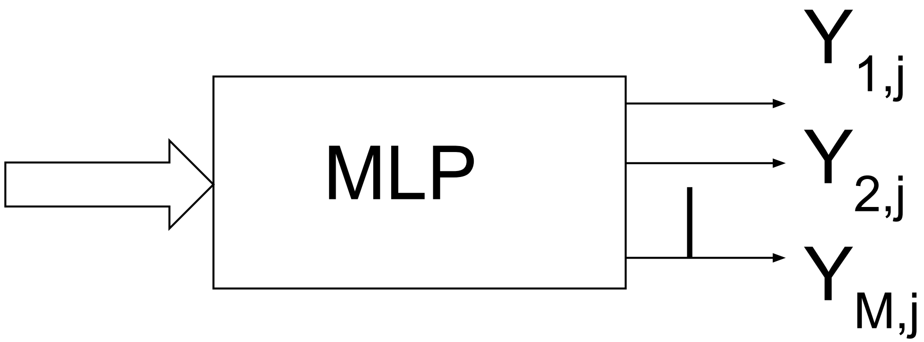
# Some heuristics (9)

- How many training data ?
  - Rule of thumb: the number of training examples should be at least five to ten times the number of weights of the network.

# Output representation and decision rule

- M-class classification problem

$Y_{k,j}(x_j)=F_k(x_j)$, $k=1,\ldots,M$



# Data representation

$$d_{k,j} = \begin{cases} 1, x_j \in C_k \\ 0, x_j \notin C_k \end{cases}$$

$$\begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \leftarrow \text{ Kth element}$$

# MLP and the a posteriori class probability

- **A multilayer perceptron classifier (using the logistic function) aproximate the a posteriori class probabilities, provided that the size of the training set is large enough.**

# The Bayes rule

- An appropriate output decision rule is the (approximate) Bayes rule generated by the a ***posteriori*** probability estimates:
- $x \in C_k$ if $F_k(x) > F_j(x)$ for all $j \neq k$

$$F(x) = \begin{bmatrix} F_1(x) \\ F_2(x) \\ \vdots \\ F_M(x) \end{bmatrix}$$