Multilayer Feedforward Networks

Berlin Chen, 2002

Introduction

- The single-layer perceptron classifiers discussed previously can only deal with linearly separable sets of patterns

- The multilayer networks to be introduced here are the most widespread neural network architecture
  - Made useful until the 1980s, because of lack of efficient training algorithms (McClelland and Rumelhart 1986)

Introduction

- Supervised Error Back-propagation Training
  - The mechanism of backward error transmission (delta learning rule) is used to modify the synaptic weights of the internal (hidden) and output layers
    - The mapping error can be propagated into hidden layers
  - Can implement arbitrary complex/output mappings or decision surfaces for to separate pattern classes
    - For which, the explicit derivation of mappings and discovery of relationships is almost impossible
  - Produce surprising results and generalizations

Linearly Non-separable Pattern Classification

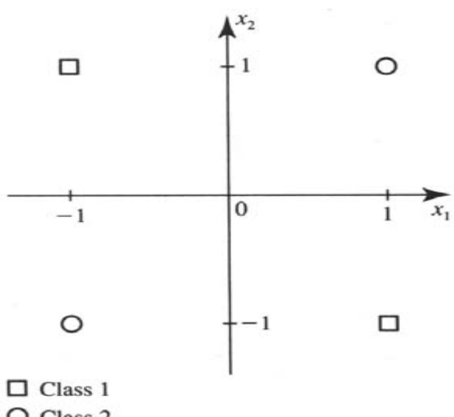
- **Linearly non-separable dichotomization**
  - For two training sets C_1 and C_2 of the augmented patterns, if no weight vector ***w*** exists such that

$$\boldsymbol{w}^t \boldsymbol{y} > 0 \quad \text{for each} \quad \boldsymbol{y} \in C_1$$

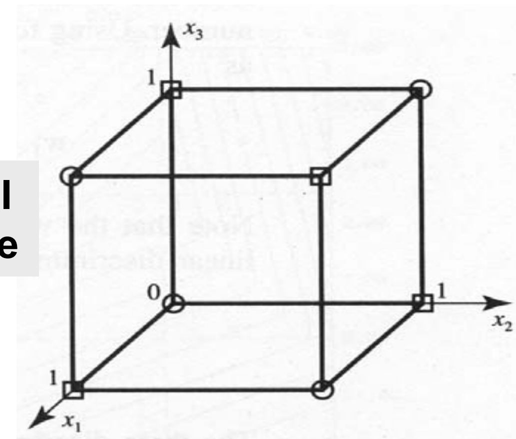
$$\boldsymbol{w}^t \boldsymbol{y} < 0 \quad \text{for each} \quad \boldsymbol{y} \in C_2$$

    - Then the patterns set C_1 and C_2 are *linearly non-separable*

Two-dimensional Input pattern space



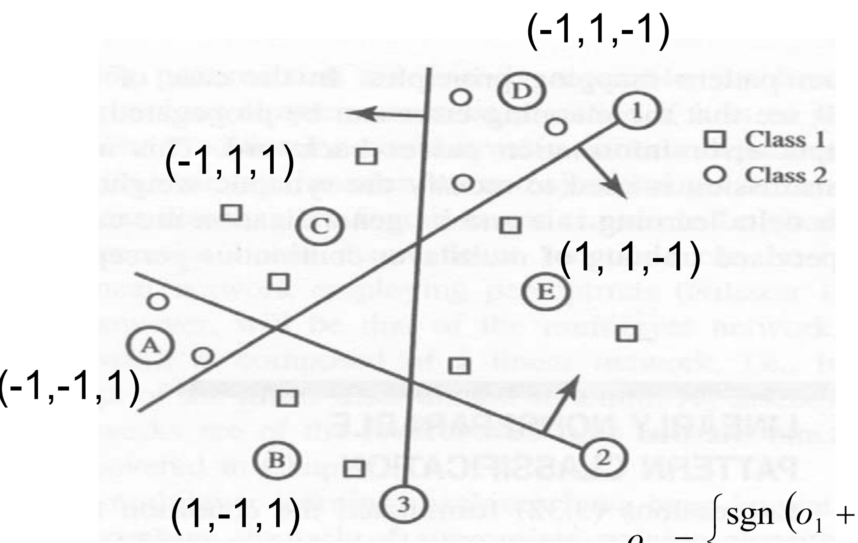
Three-dimensional Input pattern space



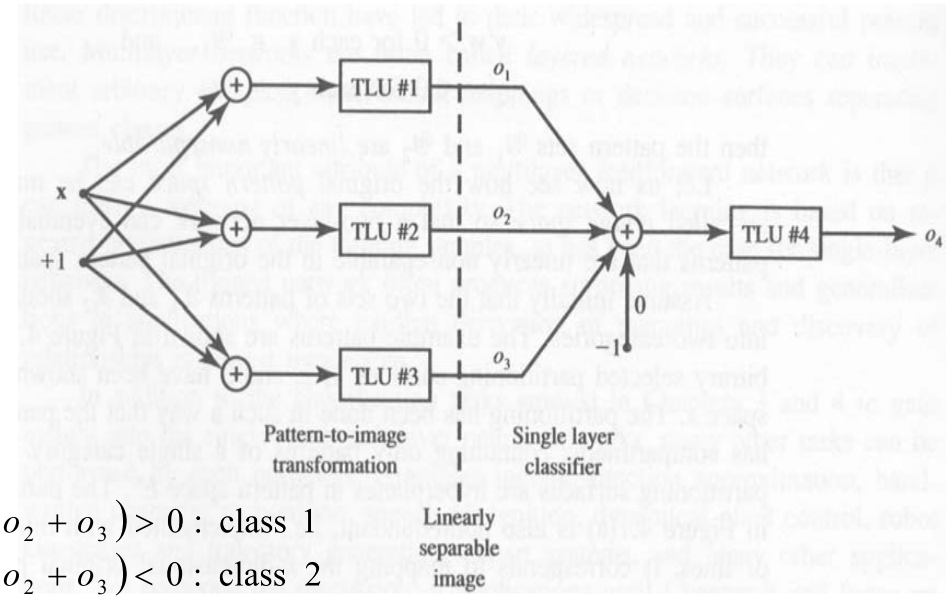
Linearly Non-separable Pattern Classification

- Map the patterns in the original ***pattern space*** into the so-called ***image space*** such that a two-layer network can classify them

2-dimensional pattern space



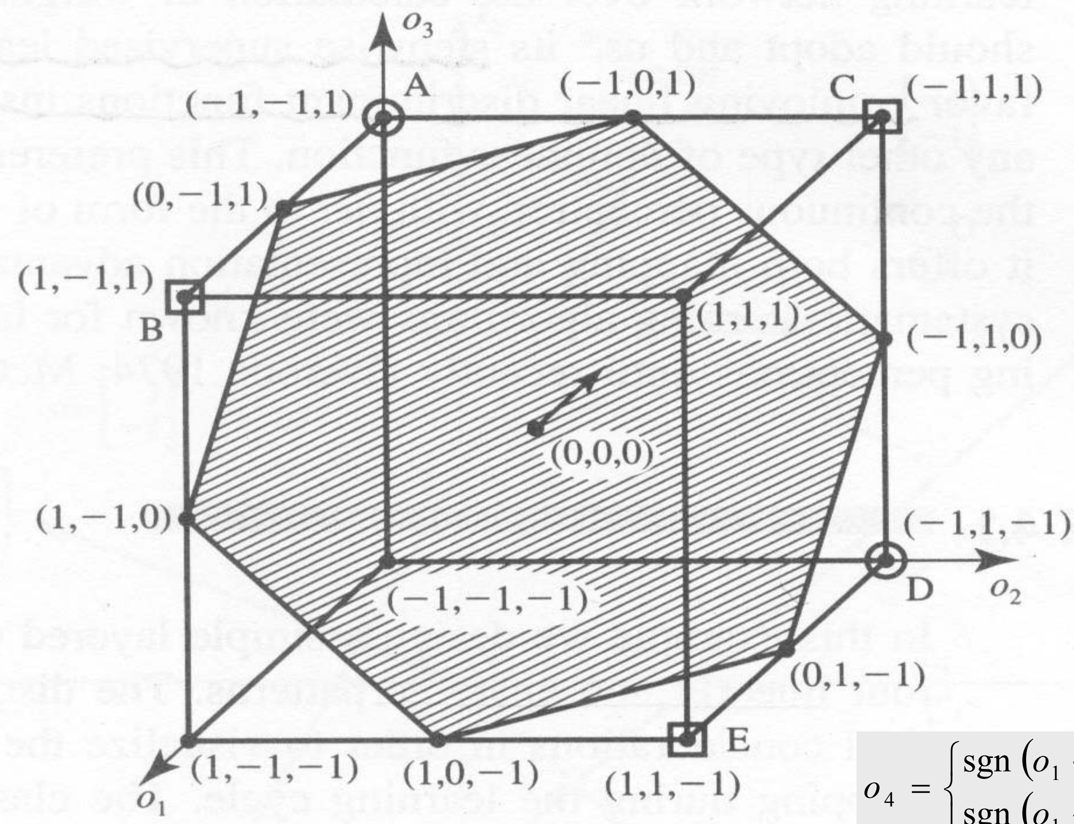
3-dimensional image space



$$o_4 = \begin{cases} \operatorname{sgn}(o_1 + o_2 + o_3) > 0 : \text{ class } 1 \\ \operatorname{sgn}(o_1 + o_2 + o_3) < 0 : \text{ class } 2 \end{cases}$$

Linearly Non-separable Pattern Classification

- Patterns mapped into the three-dimensional cube
  - Produce linearly separable images in the image space



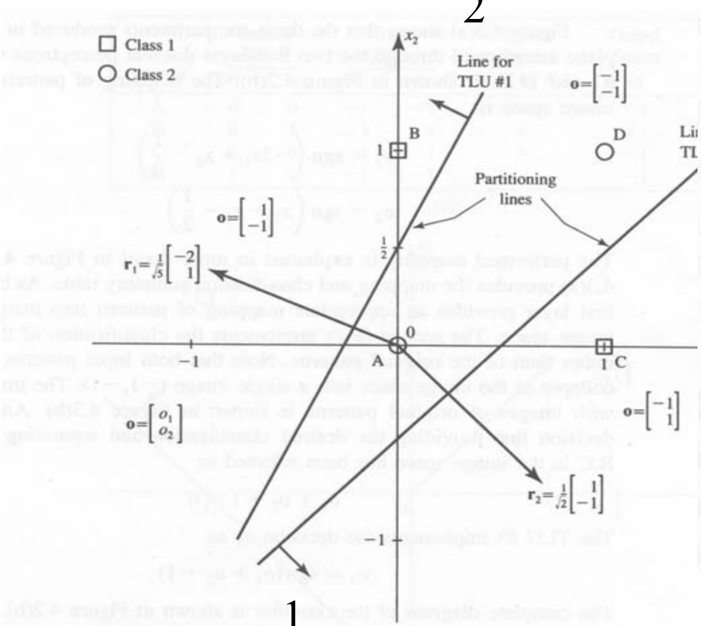
$$o_4 = \begin{cases} \operatorname{sgn}(o_1 + o_2 + o_3) > 0 : \text{ class } 1 \\ \operatorname{sgn}(o_1 + o_2 + o_3) < 0 : \text{ class } 2 \end{cases}$$

Linearly Non-separable Pattern Classification

- **Example 4.1**: the XOR function using a simple layered classifier *(with parameters produced by inspection)*

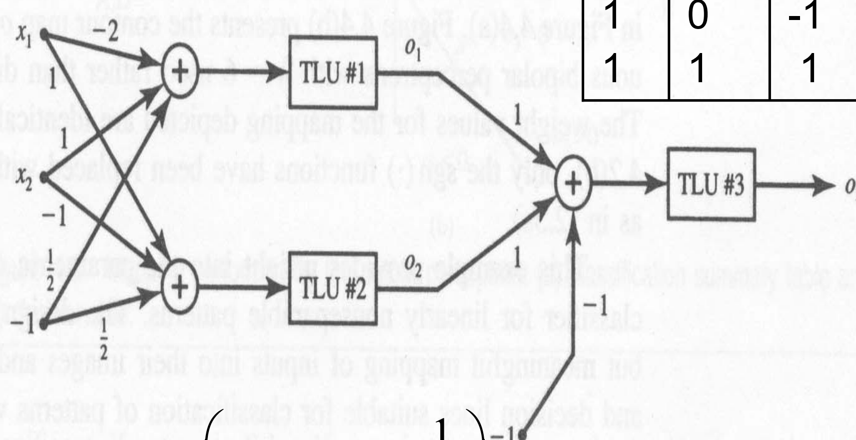
x_1	x_2	Output
0	0	1
0	1	-1
1	0	-1
1	1	1

$$-2x_1 + x_2 - \frac{1}{2} = 0$$



$$x_1 - x_2 - \frac{1}{2} = 0$$

$$o_1 = \operatorname{sgn}\left(-2x_1 + x_2 - \frac{1}{2}\right)$$



$$o_2 = \operatorname{sgn}\left(x_1 - x_2 - \frac{1}{2}\right)$$

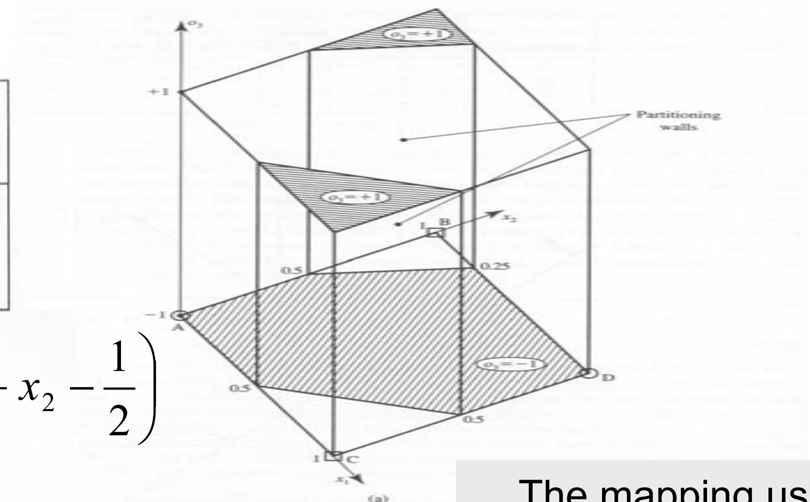
bipolar discrete perceptron

Linearly Non-separable Pattern Classification

The mapping using discrete perceptrons

- **Example 4.1**

Symbol	Pattern Space x_1	Pattern Space x_2	Image Space o_1	Image Space o_2	TLU #3 Input $o_1 + o_2 + 1$	Output Space o_3	Class Number
A	0	0	−1	−1	−	−1	2
B	0	1	1	−1	+	+1	1
C	1	0	−1	1	+	+1	1
D	1	1	−1	−1	−	−1	2



$$o_1 = \operatorname{sgn}\left(-2x_1 + x_2 - \frac{1}{2}\right)$$

$$o_2 = \operatorname{sgn}\left(x_1 - x_2 - \frac{1}{2}\right)$$

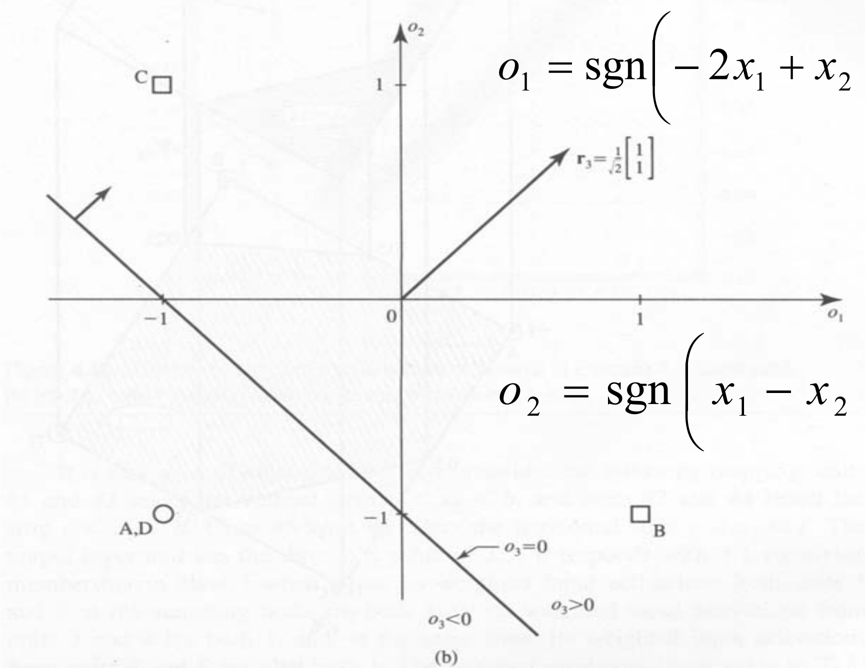
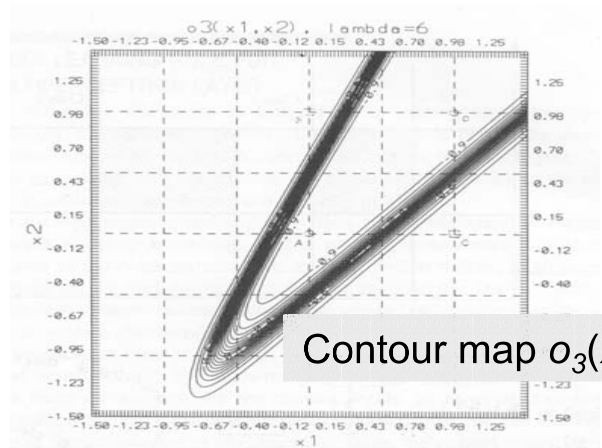


Figure 4.3 Mapping performed by the output perceptron: (a) classification summary table and (b) decision line.

The mapping using continuous perceptrons

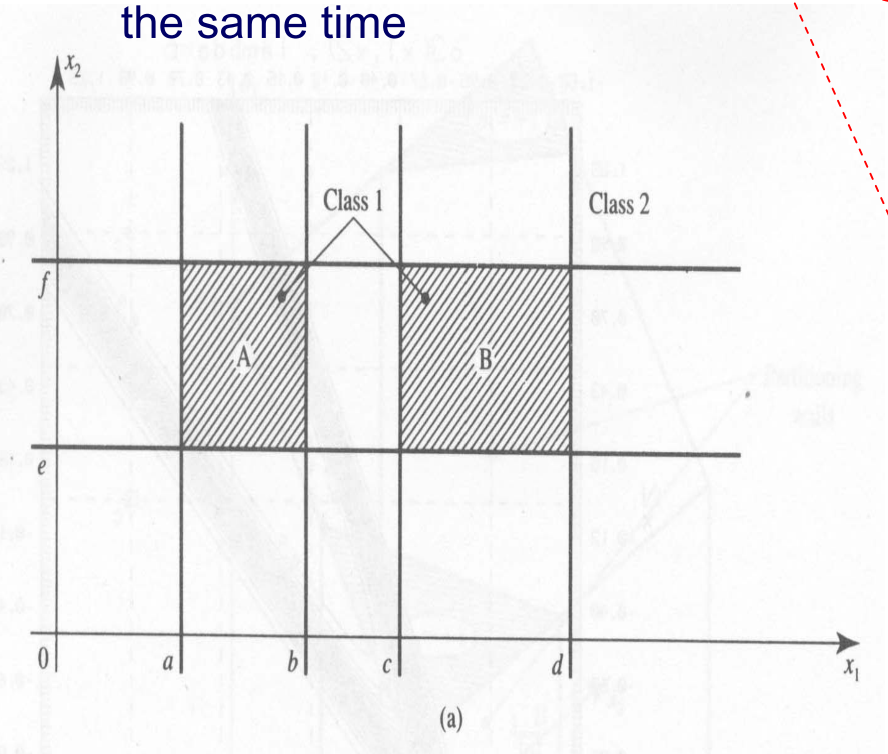


Contour map $o_3(x_1,x_2)$

Linearly Non-separable Pattern Classification

- **Another example**: classification of planner patterns

For class 1, only one set of them will be both activated at the same time



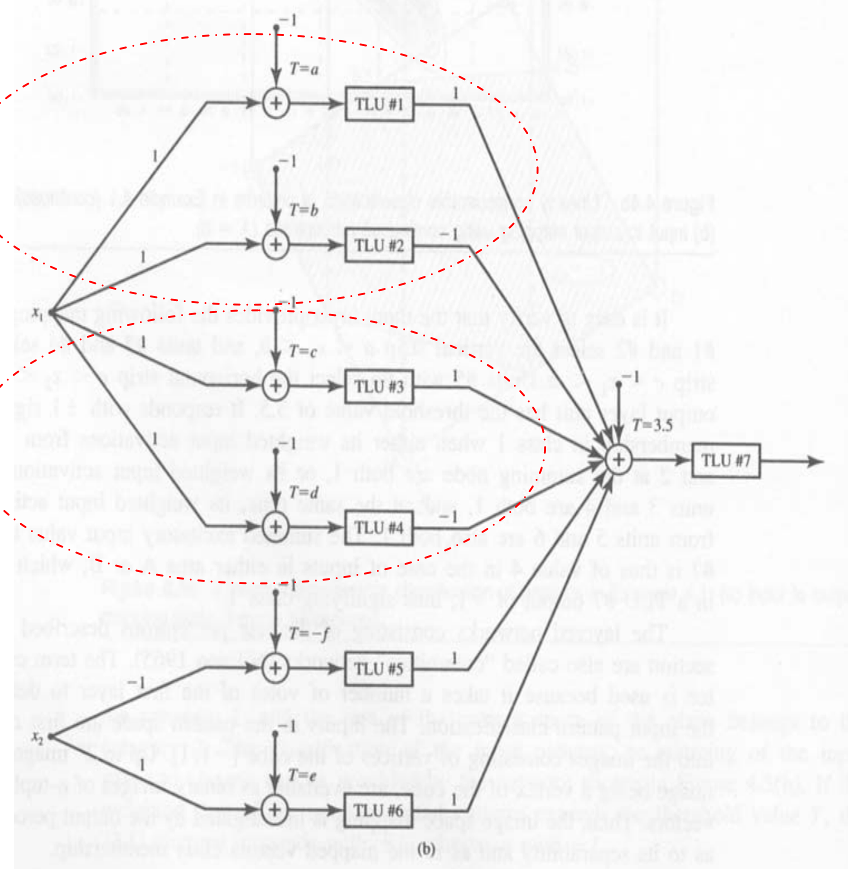


Figure 4.5 Planar pattern classification example: (a) pattern space and (b) discrete perceptron classifier network.

Linearly Non-separable Pattern Classification

- The layered networks with discrete perceptrons described here are also called “committee” network
  - Committee → Voting

Input pattern space → **Image space** → **Class membership**

$[1, -1]^N$

A vertex of cube

Error Back-propagation Training for Multi-layer Feed-forward Networks

- The error back-propagation training algorithm has reawaked the scientific and engineering community to the modeling of many quantitative phenomena using neural networks

- The ***Delta Learning Rule*** is applied
  - Each neuron has a nonlinear and differentiable activation function (sigmoid function)
  - Neurons' (synaptic) weights are adjusted based on the least mean square (LMS) criterion

Error Back-propagation Training for Multi-layer Feed-forward Networks

- **Training**: experiential acquisition of input/output mapping knowledge within multilayer networks
  - Input patterns submitted sequentially during training
  - **Synaptic weights** and **thresholds** adjusted to reduce the mean square classification error
    - The weight adjustments enforce backward from the "**output layer**" through the "**hidden layers**" toward the "**input layer**"
  - Continued until the network are within an acceptable overall error for the whole training set

Error Back-propagation Training for Multi-layer Feed-forward Networks

- Revisit the Delta Learning Rule for the single-layer network
  - Continuous activation functions $\boldsymbol{o} = \Gamma[\boldsymbol{Wy}], \ \boldsymbol{net} = \boldsymbol{Wy}$
  - Gradient descent search

The Derivation for A Specific Neuron *k*

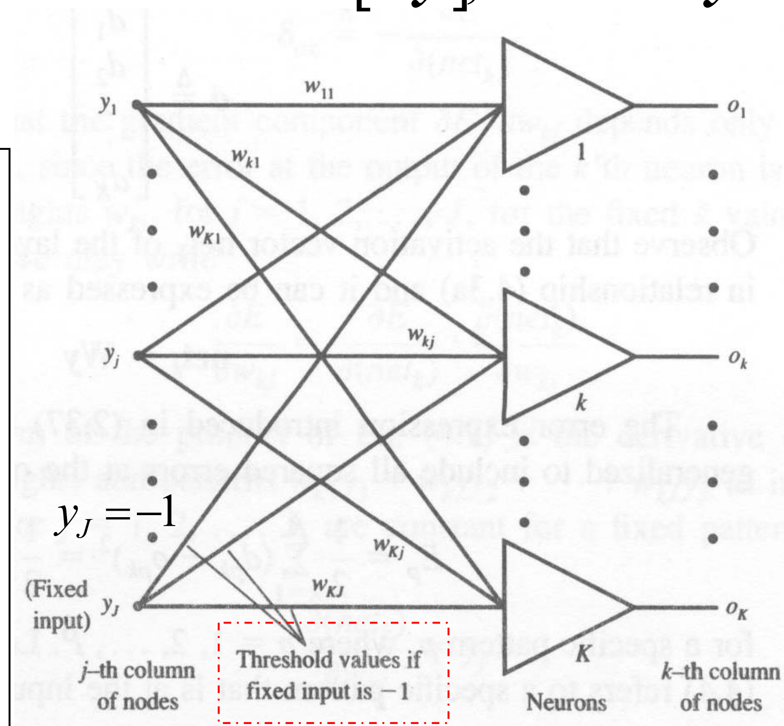
$$E = \frac{1}{2}\sum_{k=1}^{K}(d_k - o_k)^2, \quad o_k = f(net_k) = f\left(\sum_{j=1}^{J} w_{kj} y_j\right)$$

$$w_{kj} = w_{kj} + \Delta w_{kj}$$

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

negative gradient decent formula

$$= \eta (d_k - o_k) f'(net_k) y_j$$



Error Back-propagation Training for Multi-layer Feed-forward Networks

- Revisit the Delta Learning Rule for the single-layer network

The definition of the *error signal term* δ for a specific neuron k

$$\delta_{ok} \overset{\Delta}{=} -\frac{\partial E}{\partial(net_k)} \Rightarrow \delta_{ok} \overset{\Delta}{=} -\frac{\partial E}{\partial f(net_k)} \frac{\partial f(net_k)}{net_k}$$

$$\frac{\partial E}{\partial f(net_k)} = -(d_k - o_k)$$

$$\frac{\partial f(net_k)}{\partial net_k} = f'(net_k)$$

$$\frac{\partial(net_k)}{\partial w_{kj}} = \frac{\partial\left(\sum_{j=1}^{J} w_{kj} y_j\right)}{\partial w_{kj}} = y_j$$

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial(net_k)} \frac{\partial(net_k)}{\partial w_{kj}} = -\delta_{ok} y_j \Rightarrow$$

$$\Delta w_{kj} = -\eta \frac{\partial E}{\partial w_{kj}} = \eta \delta_{ok} y_j$$

$$w_{kj} = w_{kj} + \eta \delta_{ok} y_j$$

Error Back-propagation Training for Multi-layer Feed-forward Networks

- Revisit the Delta Learning Rule for the single-layer network
  - Unipolar continuous activation function

$$f(net_k) = \frac{1}{1+\exp(-net_k)} \Rightarrow f'(net_k) = o_k(1-o_k)$$

$$\Delta w_{kj} = \eta \underbrace{(d_k - o_k)o_k(1-o_k)}_{\delta_{ok}} y_j$$

  - Bipolar continuous activation function

$$f(net_k) = \frac{2}{1+\exp(-net_k)} - 1 \Rightarrow f'(net_k) = \frac{1}{2}(1-o_k^2)$$

$$\Delta w_{kj} = \eta \cdot \underbrace{\frac{1}{2}(d_k - o_k)(1-o_k^2)}_{\delta_{ok}} y_j$$

Error Back-propagation Training for Multi-layer Feed-forward Networks

- Revisit the Delta Learning Rule for the single-layer network

$$\boldsymbol{W}' = \boldsymbol{W} + \eta \boldsymbol{\delta} \boldsymbol{y}^t$$

$$\boldsymbol{W} = \begin{bmatrix} w_{11} & w_{12} & \cdots\cdots & w_{1J} \\ w_{21} & w_{22} & \cdots\cdots & w_{2J} \\ . & . & & . \\ . & . & & . \\ w_{K1} & w_{K2} & \cdots\cdots & w_{KJ} \end{bmatrix} \quad \boldsymbol{\delta} = \begin{bmatrix} \delta_{o1} \\ \delta_{o2} \\ . \\ . \\ \delta_{oK} \end{bmatrix} \quad \boldsymbol{y}^t = \left[y_1 \, y_2 \, .. \, y_J \right]$$

  - δ_{ok} are local error signals dependent only on o_k and d_k

Error Back-propagation Training for Multi-layer Feed-forward Networks

- Generalized Delta learning rule for hidden Layers

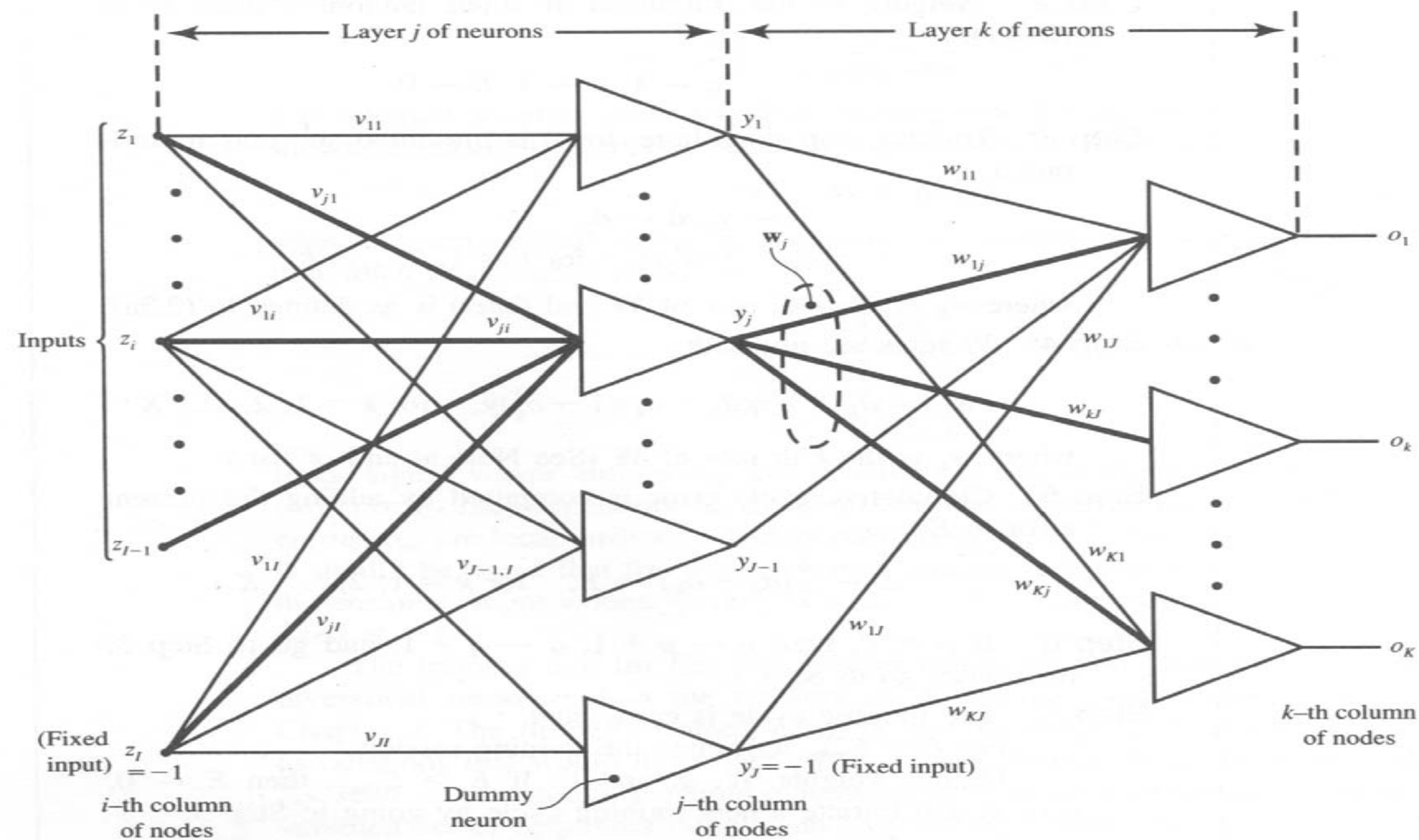


Figure 4.7 Layered feedforward neural network with two continuous perceptron layers.

Error Back-propagation Training for Multi-layer Feed-forward Networks

- Apply the negative gradient decent formula for the hidden layer

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

$$net_j = \sum_{i=1}^{I} v_{ji} z_i$$

$$\Delta v_{ji} = -\eta \frac{\partial E}{\partial net_j} \cdot \frac{\partial net_j}{\partial v_{ji}}$$

$$\delta_{yj} \overset{\Delta}{=} -\frac{\partial E}{\partial net_j}$$

The error signal term of the hidden layer having output y_j

$$\Delta v_{ji} = \eta \delta_{yj} z_i$$

Error Back-propagation Training for Multi-layer Feed-forward Networks

- Apply the negative gradient decent formula for the hidden layer

$$\delta_{yj} = -\frac{\partial E}{\partial y_j} \cdot \frac{\partial y_j}{\partial net_j}$$

$$E = E = \frac{1}{2}\sum_{k=1}^{K}[d_k - o_k]^2 = \frac{1}{2}\sum_{k=1}^{K}[d_k - f(net_k)]^2$$

$$net_k = \sum_{j=1}^{J} w_{kj} y_j$$

$$\frac{\partial E}{\partial y_j} = \frac{\partial E}{\partial net_k} \cdot \frac{\partial net_k}{\partial y_j}$$

$$\frac{\partial E}{\partial net_k} \cdot \frac{\partial net_k}{\partial y_j} = \frac{1}{2}\sum_{k=1}^{K}\left\{\frac{\partial\left([d_k - f(net_k)]^2\right)}{\partial net_k}\frac{\partial net_k}{\partial y_j}\right\}$$

$$= -\sum_{k=1}^{K}\{\underbrace{[d_k - f(net_k)]f'(net_k)}_{\delta_{ok}} w_{kj}\}$$

Error Back-propagation Training for Multi-layer Feed-forward Networks

- Generalized Delta learning rule for hidden Layers

$$y_j = f(net_j) \Rightarrow \frac{\partial y_j}{\partial net_j} = f'(net_j)$$

$$\delta_{yj} = f'(net_j)\sum_{k=1}^{K}\left[(d_k - o_k)f'(net_k)w_{kj}\right]$$

$$= f'(net_j)\sum_{k=1}^{K}\delta_{ok} w_{kj}$$

$$\Delta v_{ji} = \eta\delta_{yj} z_i$$

$$v_{ji} = v_{ji} + \Delta v_{ji}$$

$$= v_{ji} + \eta f'(net_j) z_i \sum_{k=1}^{K}\delta_{ok} w_{kj}$$

Error Back-propagation Training for Multi-layer Feed-forward Networks

- Generalized Delta learning rule for hidden Layers
  - Bipolar continuous activation function

$$v_{ji} = v_{ji} + \eta f'(net_j) z_i \sum_{k=1}^{K} \left[(d_k - o_k) f'(net_k) w_{kj} \right]$$

$$= v_{ji} + \frac{1}{2}\eta \left(1 - y_j^2\right) z_i \sum_{k=1}^{K} \left[\frac{1}{2}(d_k - o_k)\left(1 - o_k^2\right) w_{kj} \right]$$

  - Unipolar continuous activation function

$$v_{ji} = v_{ji} + \eta f'(net_j) z_i \sum_{k=1}^{K} \left[(d_k - o_k) f'(net_k) w_{kj} \right]$$

$$= v_{ji} + \eta y_j (1 - y_j) z_i \sum_{k=1}^{K} \left[(d_k - o_k) o_k (1 - o_k) w_{kj} \right]$$

The adjustment of weights leading to neuron *j* in the hidden layer is proportional to the **weighted sum** of all δ values at the adjacent following layer of nodes connecting neuron *j* with the output

Error Back-propagation Training for Multi-layer Feed-forward Networks

Bipolar continuous activation function

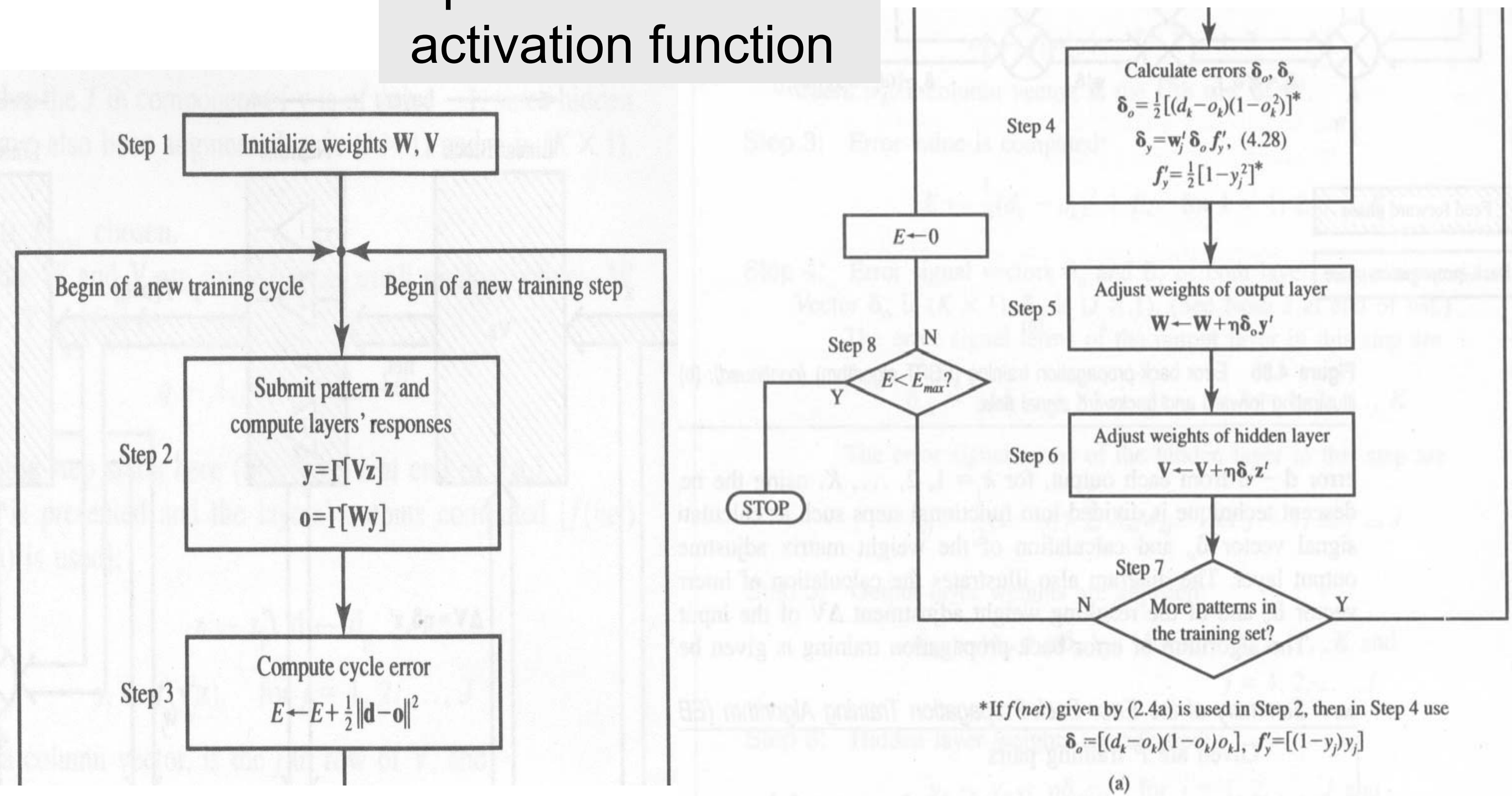
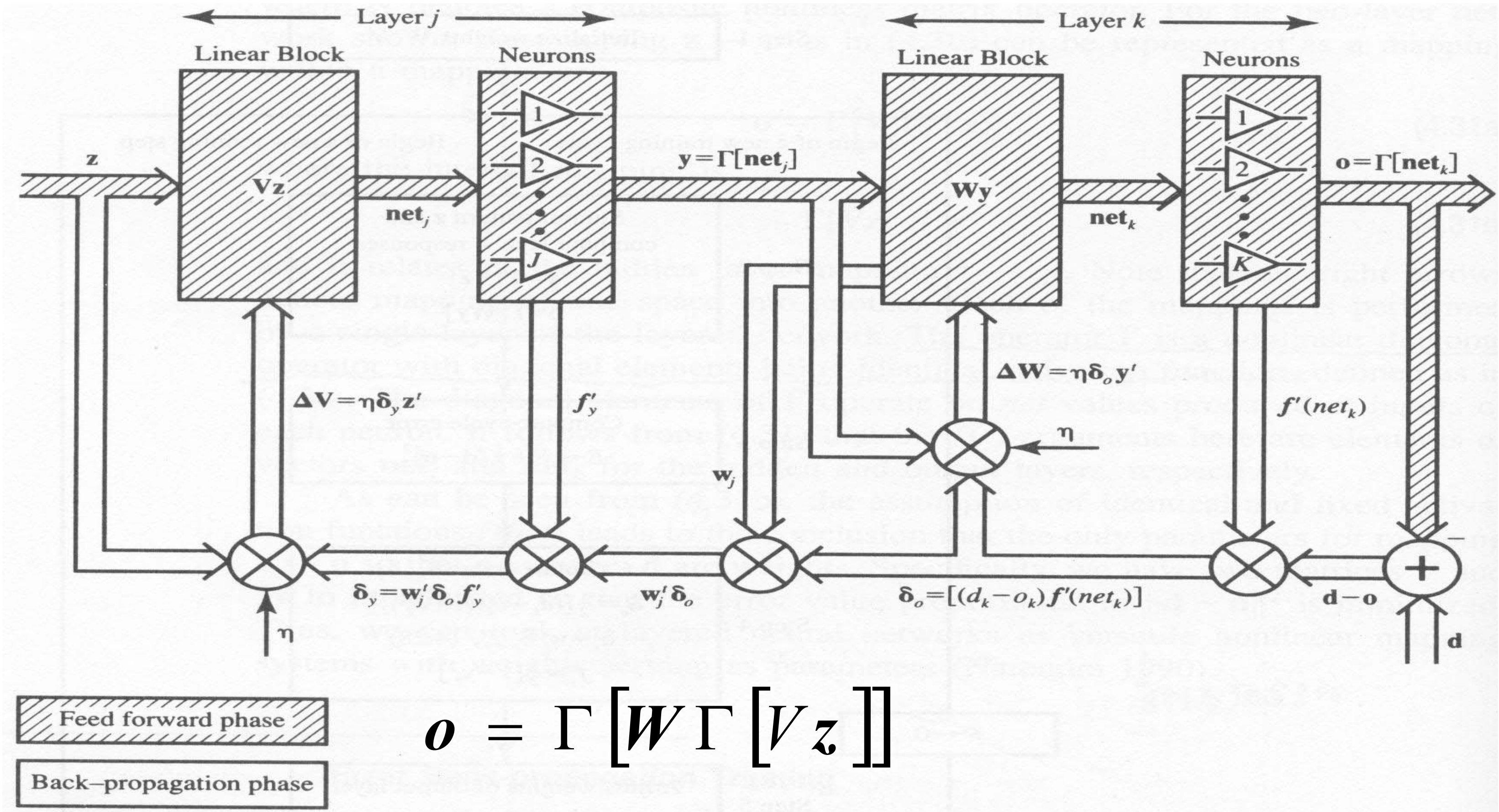


Figure 4.8a Error back-propagation training (EBPT algorithm): (a) algorithm flowchart.

Error Back-propagation Training for Multi-layer Feed-forward Networks



$$\boldsymbol{o} = \Gamma\left[\boldsymbol{W}\,\Gamma\left[\boldsymbol{V}\boldsymbol{z}\right]\right]$$

(b)

Figure 4.8b Error back-propagation training (EBPT algorithm) *(continued)*: (b) block diagram illustrating forward and backward signal flow.

Error Back-propagation Training for Multi-layer Feed-forward Networks

- The incremental learning of the weight matrix in the output and hidden layers is obtained by the outer product rule as

$$\Delta \boldsymbol{W} = \eta \boldsymbol{\delta} \boldsymbol{y}^t$$

  - Where $\boldsymbol{\delta}$ is the error signal vector of a layer and $\boldsymbol{y}$ is the input signal to that layer

- The network is nonlinear in the feedforward mode, while the error back-propagation is computed using the linearized activation
  - The slope of each neuron’s activation function

Examples of Error Back-Propagation Training

- **Example 4.2**: XOR function

o_1	o_2	Output
0	0	-1
0	1	1
1	0	1
1	1	-1

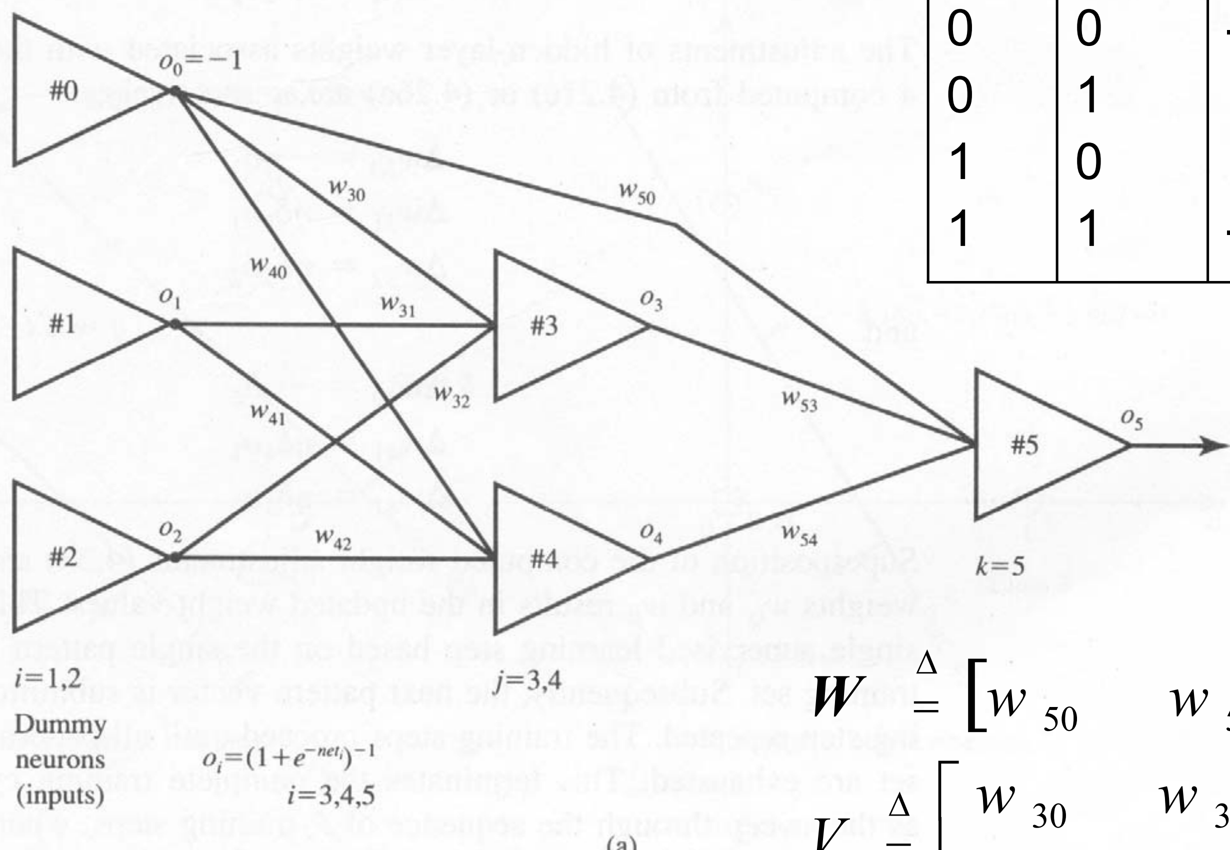


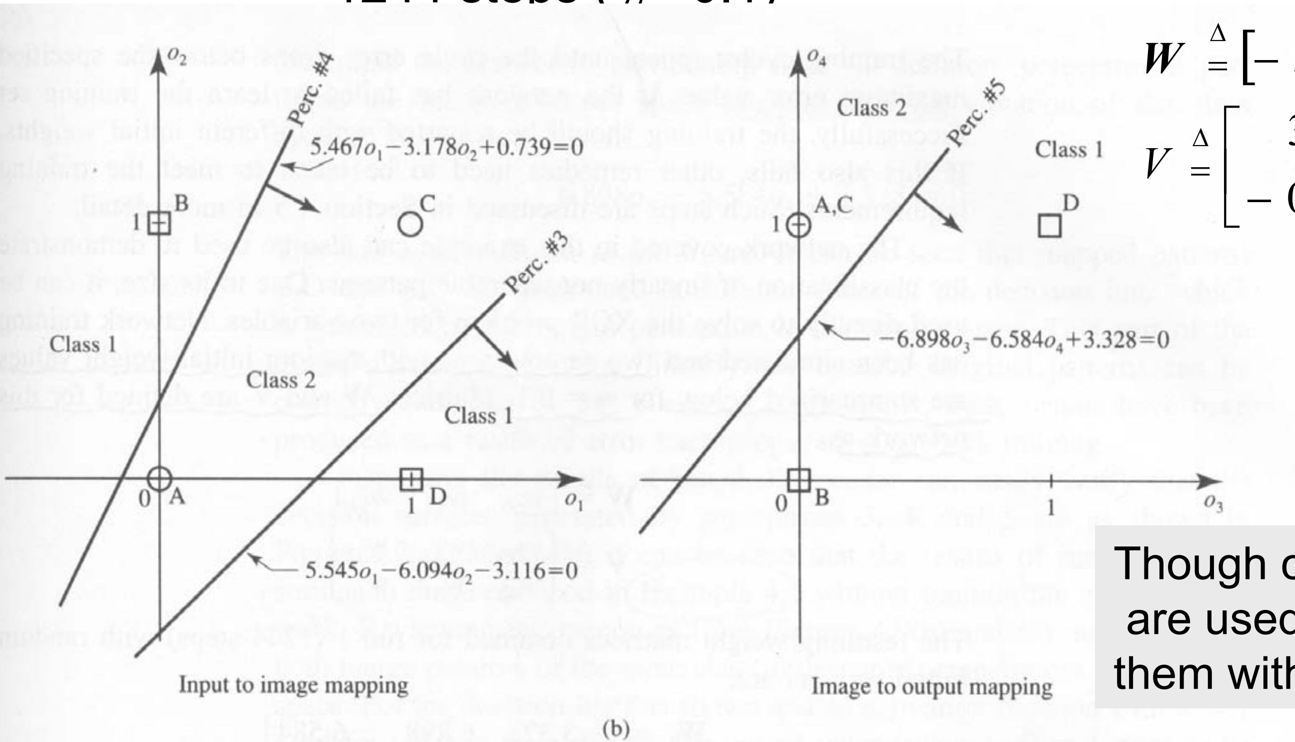
Figure 4.9a Figure for Example 4.2: (a) network diagram.

$$W \overset{\Delta}{=} \begin{bmatrix} w_{50} & w_{53} & w_{54} \end{bmatrix}$$

$$V \overset{\Delta}{=} \begin{bmatrix} w_{30} & w_{31} & w_{32} \\ w_{40} & w_{41} & w_{42} \end{bmatrix}$$

Examples of Error Back-Propagation Training

- **Example 4.2**: XOR function
  - The first sample run with random initial weight values
    - 1244 steps (η =0.1)



$$W \triangleq \begin{bmatrix} -3.328 & 6.898 & -6.584 \end{bmatrix}$$

$$V \triangleq \begin{bmatrix} 3.116 & 5.545 & -6.094 \\ -0.739 & 5.467 & 3.178 \end{bmatrix}$$

Though continuous neurons are used for training, we replace them with bipolar binary neurons

Examples of Error Back-Propagation Training

- **Example 4.2**: XOR function
  - The second sample run with random initial weight values
    - 2128 steps (η =0.1)

$$W \triangleq \begin{bmatrix} -3.967 & -8.160 & -5.376 \end{bmatrix}$$

$$V \triangleq \begin{bmatrix} 6.169 & 3.854 & 4.281 \\ -1.269 & -4.674 & -4.578 \end{bmatrix}$$

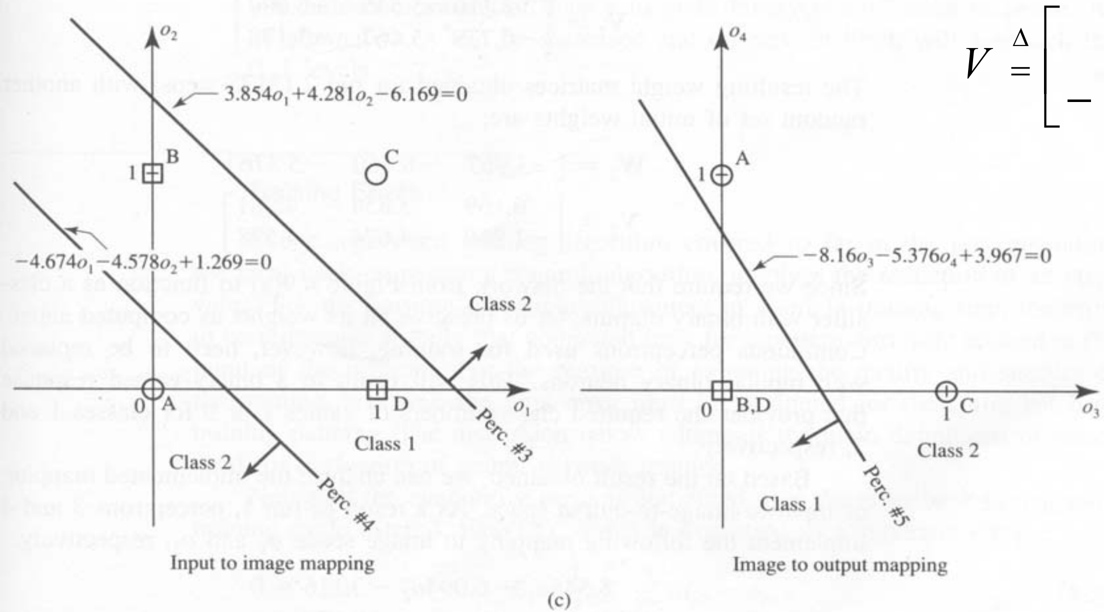


Figure 4.9b,c Figure for Example 4.2 *(continued):* (b) space transformations, run 1, and (c) space transformations, run 2.

If the network has failed to learn the training set successfully, the training should be restarted with Different initial weights

Training Errors

- For the purpose of assessing the quality and success of training, the joint error (cumulative error) must be computed for the entire batch of training patterns

$$E = \frac{1}{2} \sum_{p=1}^{P} \sum_{k=1}^{K} \left(d_{pk} - o_{pk} \right)^2$$

  - It is not very useful for comparison of networks with different numbers of training patterns and having different number of output neurons
    - *Root-mean-square normalized error*

$$E_{\text{rms}} = \frac{1}{PK} \sqrt{\sum_{p=1}^{P} \sum_{k=1}^{K} \left(d_{pk} - o_{pk} \right)^2}$$

Training Errors

- For some classification applications
  - The desired outputs below a threshold will be set to 0, while the desired outputs higher than an other threshold will be set to 1

$$o_{pk} < 0.1 \Rightarrow o_{pk} = 0$$
$$o_{pk} > 0.9 \Rightarrow o_{pk} = 1$$

  - In such cases, the **decision error** will more adequately reflects the accuracy of neural network classifiers

$$E_d = \frac{N_{err}}{PK}$$

Average number of bit errors

  - The networks in classification applications may exhibit zero decision errors while still yielding substantial E and E_{rms}

Multilayer Feedforward Networks as Function Approximators

- **Example**: a function *h(x) approximated by H(**w**,x)*

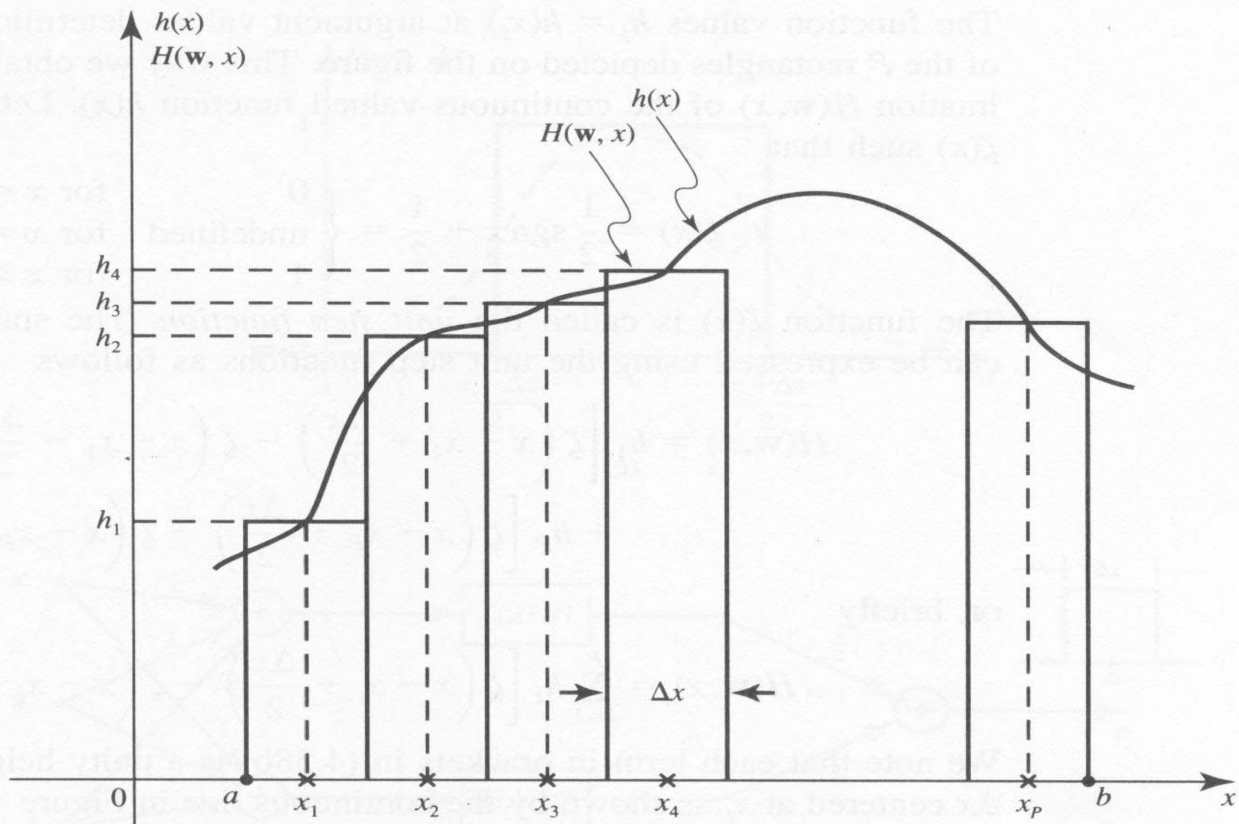


Figure 4.10 Approximation of $h(x)$ with staircase function $H(\mathbf{w}, x)$.

Multilayer Feedforward Networks as Function Approximators

- There are P samples $\{x_1, x_2, \ldots., x_p\}$, which are examples of function values in the interval (*a*, *b*)

$$x_{i+1} - x_i = \Delta x = \frac{b-a}{P}, \text{ for } i = 1, \ldots., P$$

  – Each subinterval with length Δx is

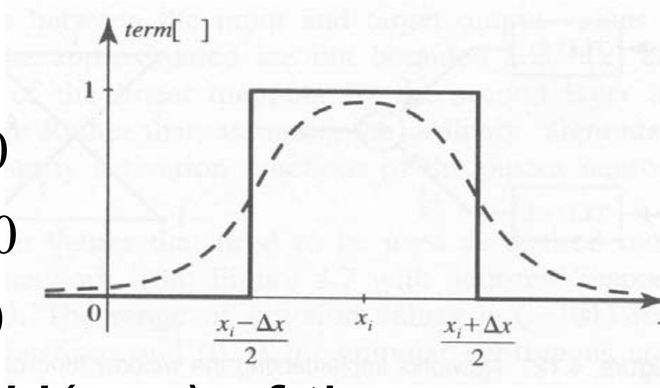
$$\left(x_i - \frac{\Delta x}{2}, x_i + \frac{\Delta x}{2} \right), \quad i = 1, 2, \ldots, \; P$$

$$x_1 - \frac{\Delta x}{2} = a, \quad x_P + \frac{\Delta x}{2} = b$$

Multilayer Feedforward Networks as Function Approximators

- Define a unit step function

$$\zeta(x) = \frac{1}{2}\mathrm{sgn}(x) + \frac{1}{2} = \begin{cases} 0 & \text{for } x < 0 \\ \text{undefined} & \text{for } x = 0 \\ 1 & \text{for } x > 0 \end{cases}$$



- Use a staircase approximation *H(**w**,x)* of the continuous-valued function *h(x)*

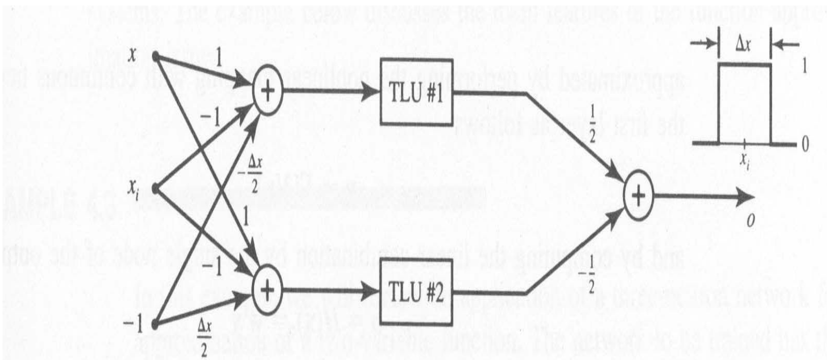
$$H(\mathbf{w}, x) = h_1\left[\zeta\left(x - \left(x_i - \frac{\Delta x}{2}\right)\right) - \zeta\left(x - \left(x_i - \frac{\Delta x}{2}\right)\right)\right]$$

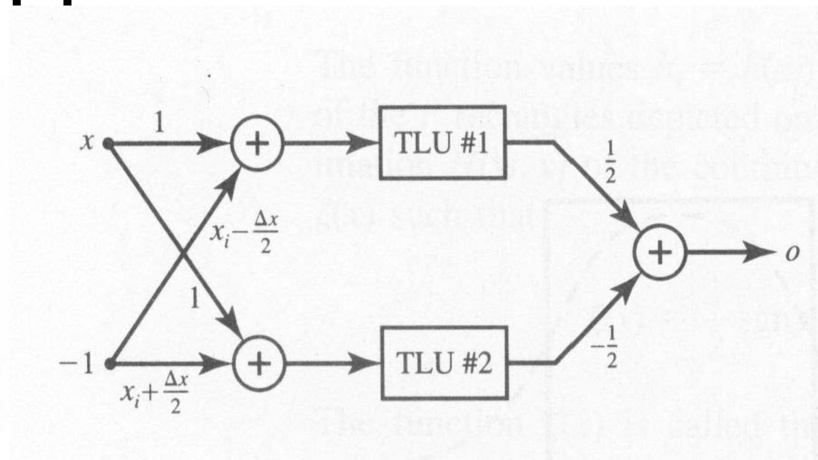
$$+ \ldots\ldots\ldots\ldots$$

$$+ h_P\left[\zeta\left(x - \left(x_P - \frac{\Delta x}{2}\right)\right) - \zeta\left(x - \left(x_P - \frac{\Delta x}{2}\right)\right)\right]$$

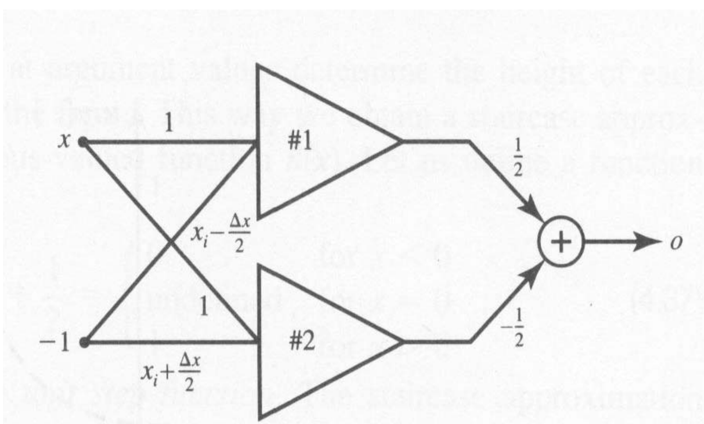
- The network will have 2P binary (nonlinear) perceptrons with TLUs in the input layer

Multilayer Feedforward Networks as Function Approximators

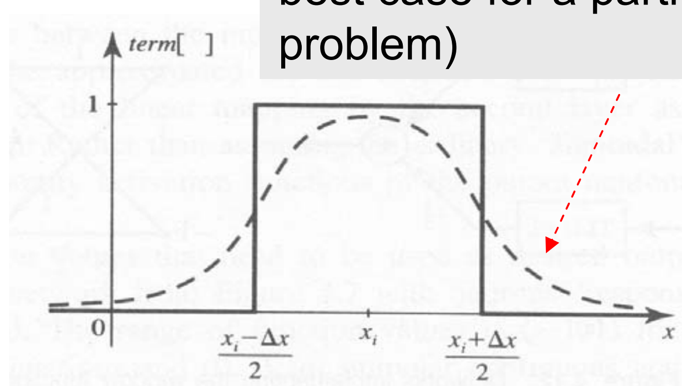




- If we replace the TLUs with continuous activation functions



bump function (may not the best case for a particular problem)



Multilayer Feedforward Networks as Function Approximators

- The output layer in the above example also can be replaced with a preceptron with nonlinear activation function

- Such a network architecture can approximate virtually any multivariable function, if provided sufficiently many hidden neurons are available

Learning Factors

- Error Curve

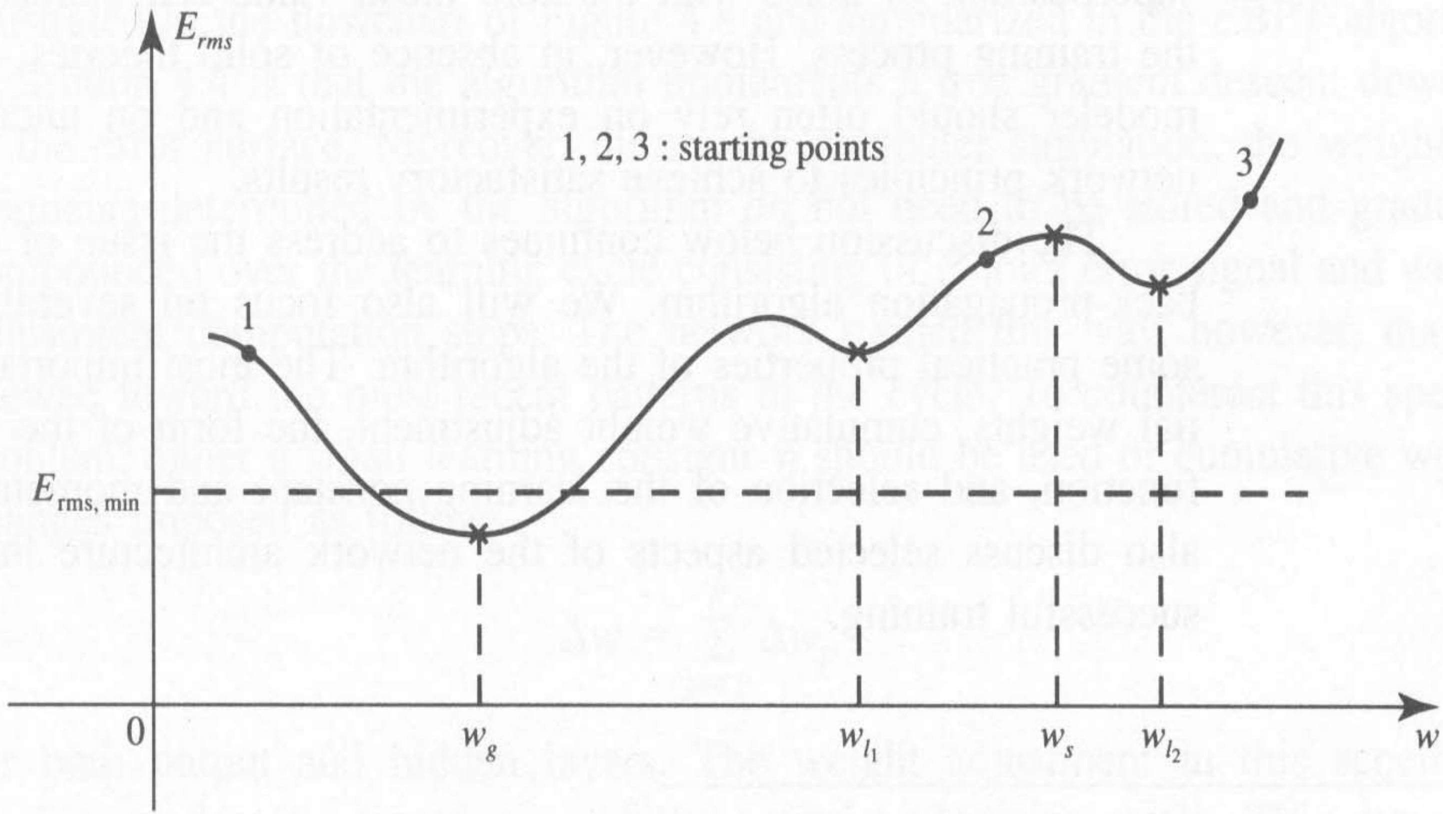


Figure 4.16 Minimization of the error E_{rms} as a function of single weight.

Learning Factors

- Initial Weights
  - The weights of the network are typically initialized at small random values
    - The initialization strongly affects the ultimate solution
    - Equal initial weights ?
  - Select another set of initial weights, and then restart !

- Incremental Updating versus Cumulative Weight Adjustment
  - **Incremental Updating**:
    - Weight adjustments do not need to be stored
    - May skewed toward the most recent patterns in the training cycle

Learning Factors

- Incremental Updating versus Cumulative Weight Adjustment
  - **Cumulative Weight Adjustment** :

$$\Delta w = \sum_{p=1}^{P} \Delta w_p$$

    - Provided that the learning constant is small enough, the cumulative weight adjustment procedure can still implement the algorithm close to the gradient decent minimization
  - *We may present the training examples in random in each training cycle*

Learning Factors

- Steepness of the activation function

$$f(net) \stackrel{\Delta}{=} \frac{2}{1 + \exp(-\lambda net)} - 1$$

$$f'(net) = \frac{2\lambda \exp(-\lambda net)}{[1 + \exp(-\lambda net)]^2}$$

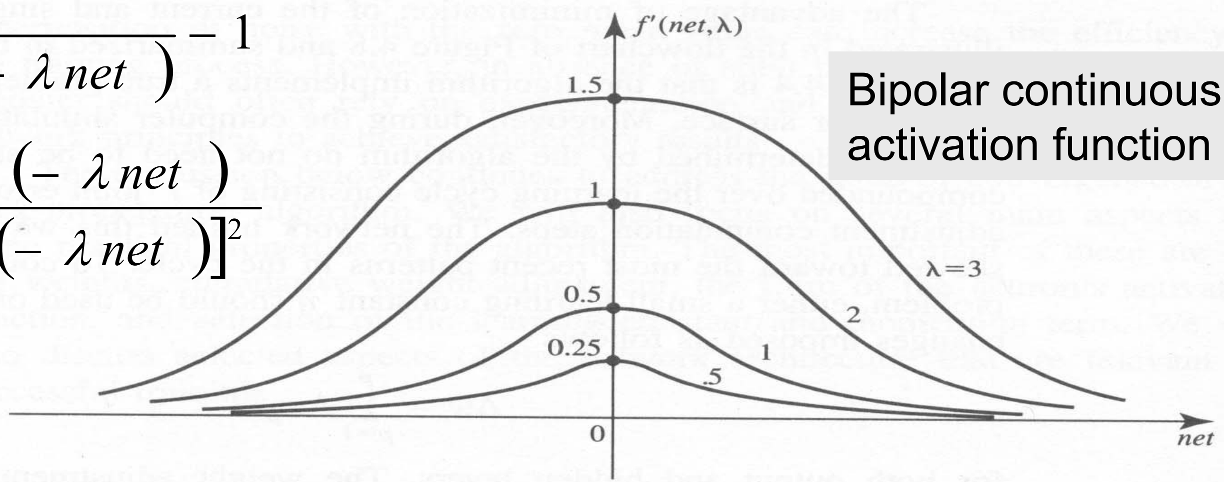


Figure 4.17 Slope of the activation function for various λ values.

Bipolar continuous activation function

  - Have a maximum value of $\frac{1}{2}\lambda$ at *net*=0
  - The large λ may yield results similar to that of large learning constant η

Learning Factors

- Momentum Method
  - Supplement the current weight adjustments with a fraction of the most recent weight adjustment

$$\Delta w(t) = -\eta \nabla E(t) + \alpha \Delta w(t)$$

- After a total of N steps with the momentum method

$$\Delta w(t) = -\eta \sum_{n=0}^{N} \alpha^{n} \nabla E(t-n)$$

Learning Factors

- Momentum Method

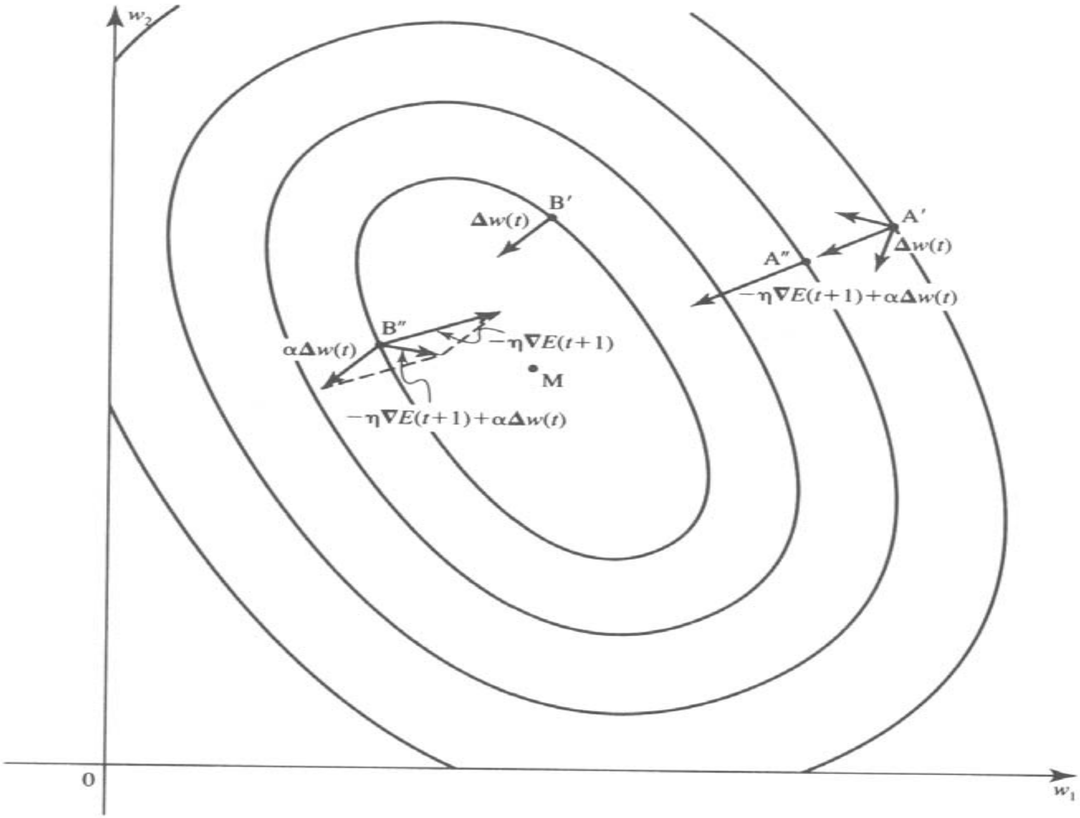


Figure 4.18 Illustration of adding the momentum term in error back-propagation training for a two-dimensional case.

Summary of Error Back-propagation Network

- A set of P training pairs $(\boldsymbol{z}_p, \boldsymbol{d}_p)$

$$\left\{ \left(\boldsymbol{z}_p, \boldsymbol{d}_p \right), \; p = 1, 2, \ldots, P \right\}$$

- Minimize the vector of total error

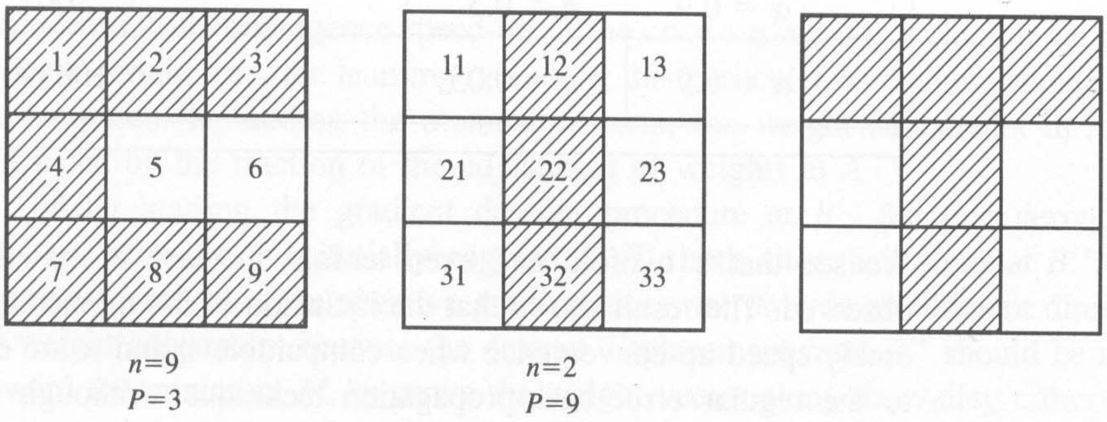
$$E = \sum_{p=1}^{P} \left\| \boldsymbol{o}\left(\boldsymbol{W}, \boldsymbol{V}, \boldsymbol{z}_p \right) - \boldsymbol{d}_p \right\|^2$$

Network Architecture vs. Data Representation

$$\mathbf{x}_1 = \begin{bmatrix}1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1\end{bmatrix}^t : \text{class C}$$

$$\mathbf{x}_2 = \begin{bmatrix}0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0\end{bmatrix}^t : \text{class I}$$

$$\mathbf{x}_3 = \begin{bmatrix}1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0\end{bmatrix}^t : \text{class T}$$



$$\boldsymbol{x}_1 = \begin{bmatrix}1 & 1\end{bmatrix}^t : \text{class C, T}$$

$$\boldsymbol{x}_2 = \begin{bmatrix}1 & 2\end{bmatrix}^t : \text{class C, I, T}$$

....

$$\boldsymbol{x}_9 = \begin{bmatrix}3 & 3\end{bmatrix}^t : \text{class C, I, T}$$

Necessary Number of Hidden Neurons

- For two-layer feedforward network
  - if the n-dimensional nonargumented input space is linear separable into M disjoint regions, the necessary number of hidden neuons would be J

$$M = 2^J$$

Mirchandini and Cao (1989)

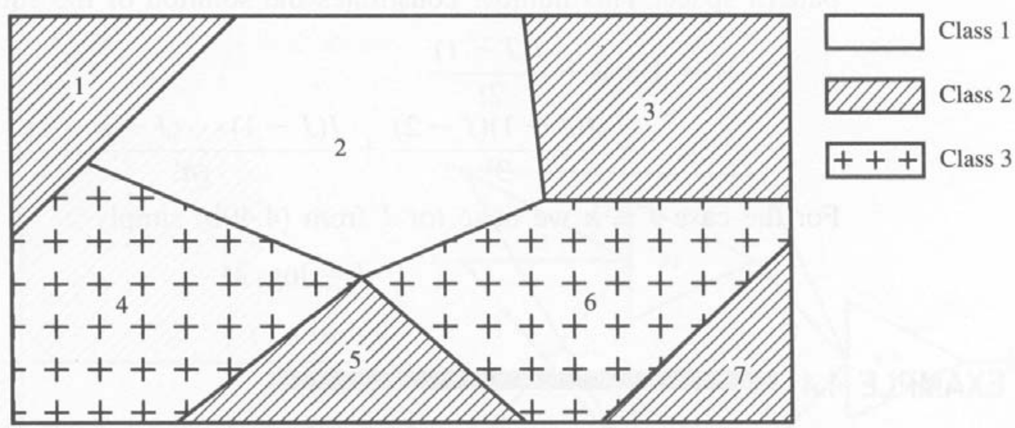


Figure 4.20 Two-dimensional input space with seven separable regions assigned to one of three classes.

Character Recognition Application

- Project a point of the character into its three closest vertical, horizontal, and diagonal bars
  - Then normalized the bar values to be between 0 and 1
  - Input vector is 13-dimensional and the activation function is unipolar continuous function

90~95% accuracy

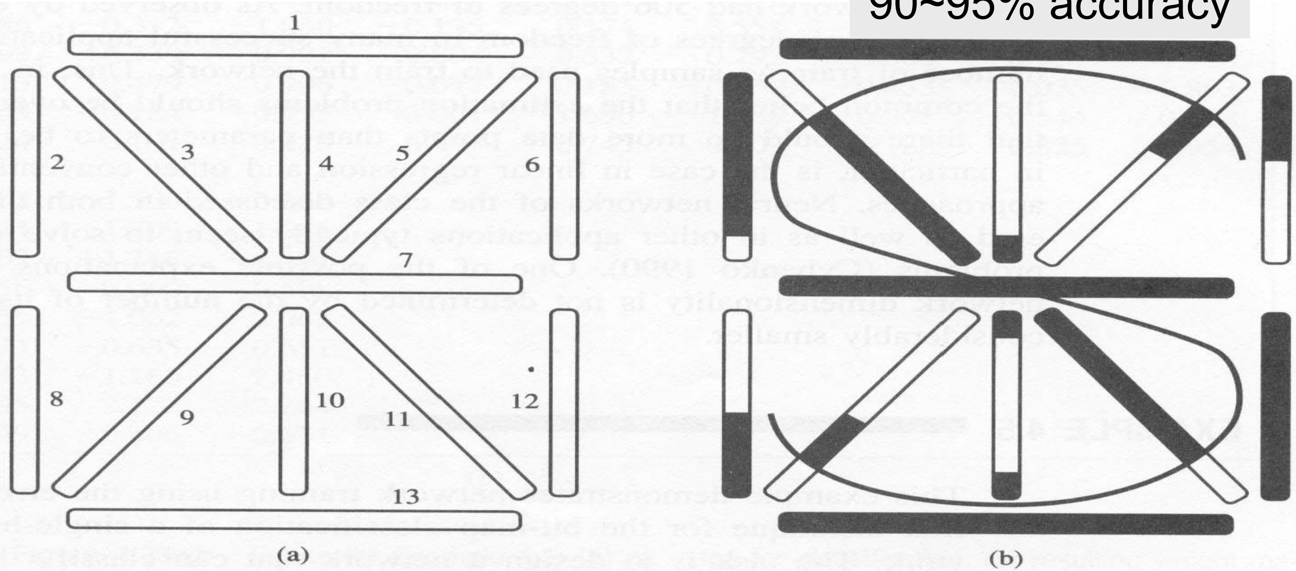
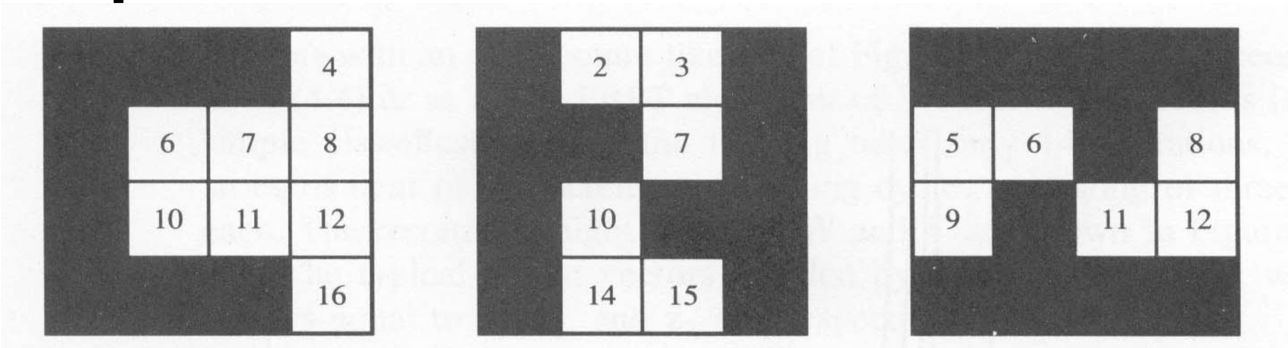


Figure 4.22 Thirteen-segment bar mask for encoding alphabetic capital letters: (a) template and (b) encoded S character. [Adapted from Burr (1988). © IEEE; reprinted with permission.]

I	J	K
14	12	26
	16	
	20	
	24	

Character Recognition Application

- **Example 4.5**



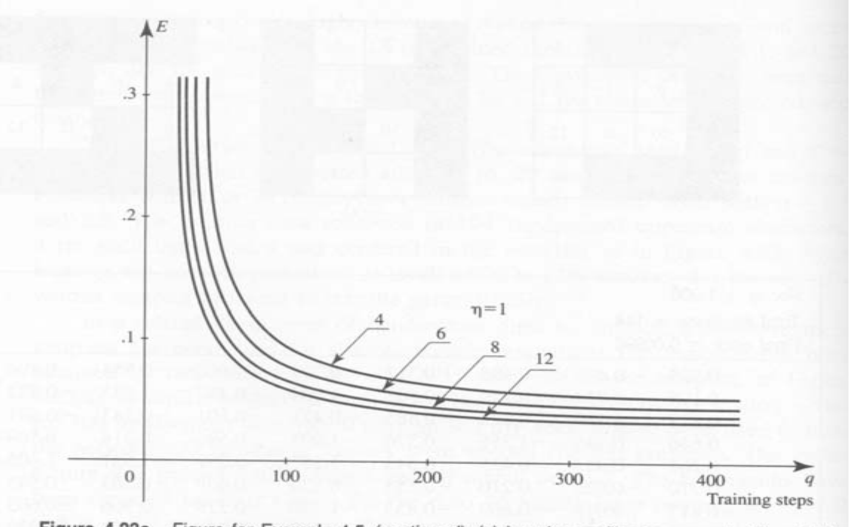
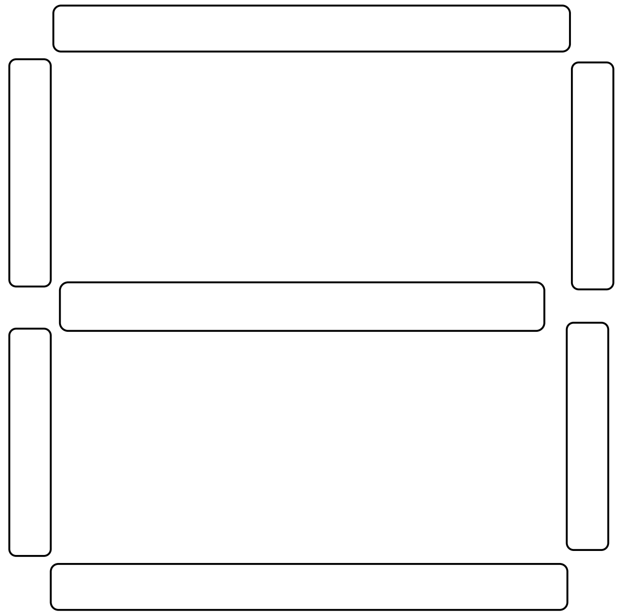


Figure 4.23c Figure for Example 4.5 *(continued):* (c) learning profiles for several different hidden layer sizes.

Digit Recognition Application

96~98% accuracy



Expert System Applications

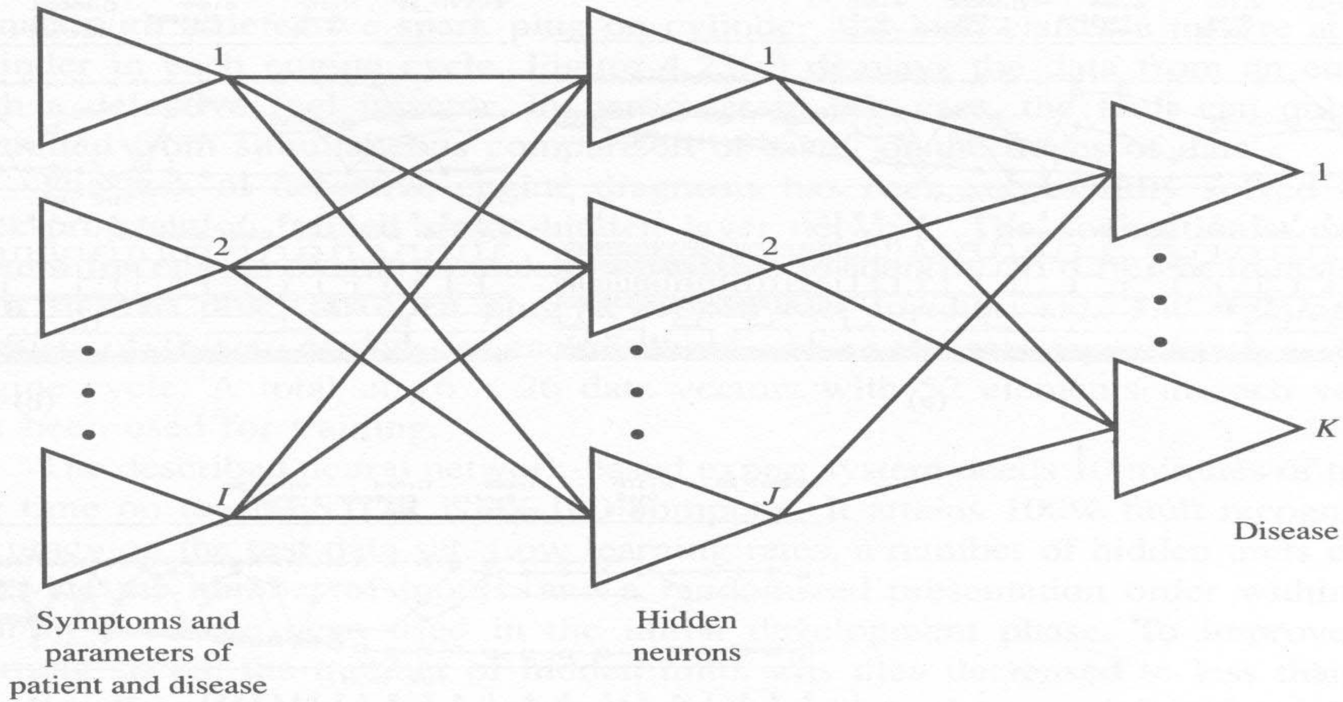
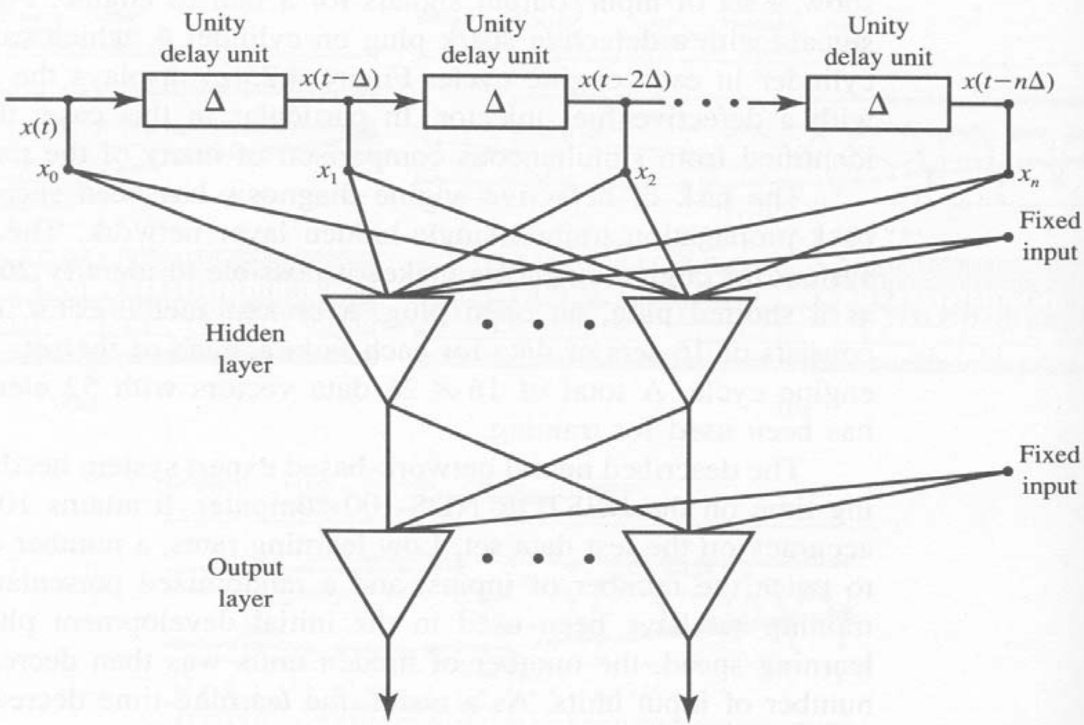


Figure 4.24 Connectionist expert system for diagnosis.

Explanation function: Neural network expert systems are typically unable to provide the user with the reasons for the decisions made.

Learning Time Sequences



Note: Δ is equal to the sampling period

Figure 4.26 A time-delay neural network converting a data sequence into the single data vector (single variable sequence shown).

Functional Link Network

- Enhance the representation of the input data

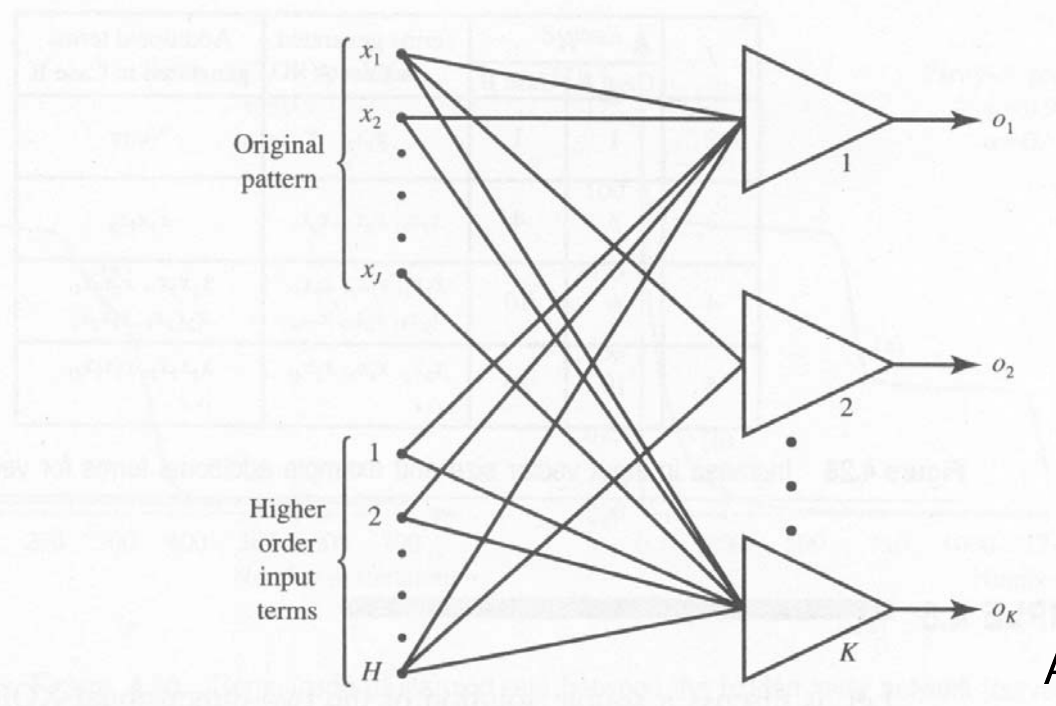


Figure 4.27 Functional link network.

J	H Case A	H Case B	Terms generated in Case A	Additional terms generated in Case B
2	1	1	x_1x_2	None
3	3	4	x_1x_2, x_1x_3, x_2x_3	$x_1x_2x_3$
4	6	10	$x_1x_2, x_1x_3, x_1x_4, x_2x_3, x_2x_4, x_3x_4$	$x_1x_2x_3, x_1x_3x_4, x_2x_3x_4, x_1x_2x_4$
5	10	20	$x_1x_2, x_1x_3, x_1x_4,$ …	$x_1x_2x_3, x_1x_3x_4,$ …

Figure 4.28 Increase in input vector size and example additional terms for vector input patterns.

Any two elements

Any three elements