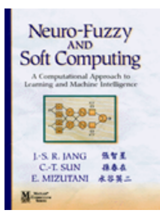
# Chapter 9: Supervised Learning Neural Networks

- Introduction (9.1)
- Perceptrons (9.2)
- Adaline (9.3)
- Backpropagation Multilayer Perceptrons (9.4)
- Radial Basis Function Networks (9.5)




Jyh-Shing Roger Jang et al., *Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence*, First Edition, Prentice Hall, 1997

## Introduction (9.1)

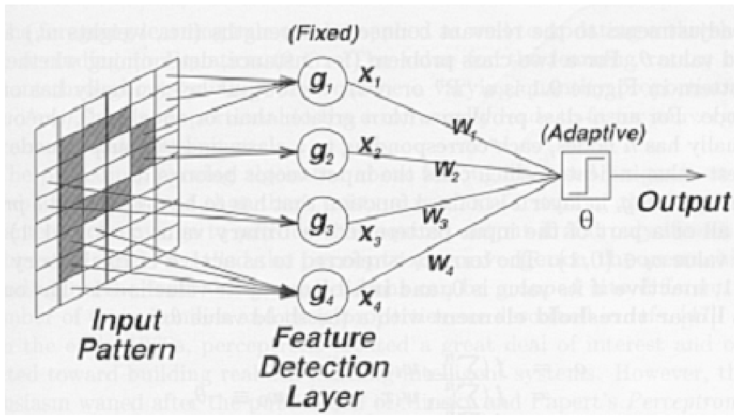
- Artificial Neural Networks (NN) have been studied since 1950
- Minsky & Papert in their report of perceptron (Rosenblatt) expressed pessimism over multilayer systems, the interest in NN dwindled in the 1970's
- The work of Rumelhart, Hinton, Williams & others, in learning algorithms created a resurgence of the lost interest in the field of NN

## Introduction (9.1) (cont.)

- Several NN have been proposed & investigated in recent years
  - Supervised versus unsupervised
  - Architectures (feedforward vs. recurrent)
  - Implementation (software vs. hardware)
  - Operations (biologically inspired vs. psychologically inspired)
- In this chapter, we will focus on modeling problems with desired input-output data set, so the resulting networks must have adjustable parameters that are updated by a supervised learning rule

# Perceptrons (9.2)

- Architecture & learning rule
  - The perceptron was derived from a biological brain neuron model introduced by Mc Culloch & Pitts in 1943
  - Rosenblatt designed the perceptron with a view toward explaining & modeling pattern recognition abilities of biological visual systems
  - The following figure illustrate a two-class problem that consists of determining whether the input pattern is a “p” or not




The perceptron

## Perceptrons (9.2) (cont.)

$$f\left(\sum_{i=1}^{i=n} w_i x_i - \theta\right) = \textbf{Output}$$

$$= f\left(\sum_{i=1}^{i=n} w_i x_i + w_o\right), w_o \equiv -\theta$$

$$= f\left(\sum_{i=1}^{i=n} w_i x_i\right), \; x_o \equiv 1$$

– A signal xi is binary, it could be active (or excitatory) if its value is 1, inactive if its value is 0 and inhibitory is its value is -1

## Perceptrons (9.2) (cont.)

- Architecture & learning rule (cont.)
  - The output unit is a linear threshold element with a threshold value $\theta$
  - wi is a modifiable weight associated with an incoming signal xi
  - The threshold $\theta$ = w0 can be viewed as the connection weight between the output unit & a dummy incoming signal x0 = 1

## Perceptrons (9.2) (cont.)

- Architecture & learning rule (cont.)
  - f(.) is the activation function of the perceptron & is either a signum function sgn(x) or a step function step (x)

$$\mathbf{sgn(x)} = \begin{cases} \mathbf{1} & \mathbf{if\ x > 0} \\ \mathbf{-1} & \mathbf{otherwise} \end{cases}$$

$$\mathbf{step(x)} = \begin{cases} \mathbf{1} & \mathbf{if\ x > 0} \\ \mathbf{0} & \mathbf{otherwise} \end{cases}$$

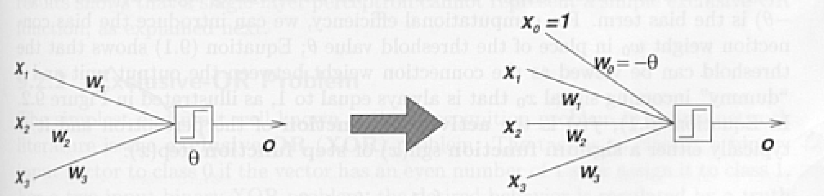



**Figure 9.2.** *Introduction of the bias connection weight; the bias term $w_0$ $(= -\theta)$ can be viewed as the connection weight between the output unit and a "dummy" incoming signal $x_0$ that is always equal to 1.*

## Perceptrons (9.2) (cont.)

- Algorithm (Single-layer perceptron)

```
1. Start with a set of random connection
   weights
2. Select an input vector x from the
   training data set
   If the perceptron provides a wrong
   response then modify all connection
   weights wi to wi = ηtixi
   where: ti is a target output
            η is a learning rate
1. Test the weight convergence: if converge
   stop else go to 1
```

This learning algorithm is based on gradient descent

## Perceptrons (cont.)

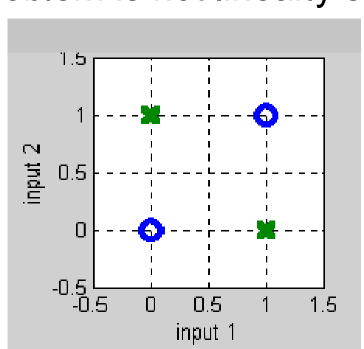
- Exclusive-OR problem (XOR)

  Goal: classify a binary input vector to class 0 if the vector has an even number of 1's, otherwise assign it to class 1

| | X | Y | Class |
|---|---|---|---|
| Desired i/o pair 1 | 0 | 0 | 0 |
| Desired i/o pair 2 | 0 | 1 | 1 |
| Desired i/o pair 3 | 1 | 0 | 1 |
| Desired i/o pair 4 | 1 | 1 | 0 |

## Perceptrons (9.2) (cont.)

- Exclusive-OR problem (XOR) (cont.)
  - From the following figure, we can say that the XOR problem is not linearly separable



## Perceptons (9.2) (cont.)

– Using a single-layer perceptron and the step function to solve this problem requires satisfying the following inequalities

$$0 * w1 + 0 * w2 + w0 \leq 0 \Leftrightarrow w0 \leq 0$$
$$0 * w1 + 1 * w2 + w0 > 0 \Leftrightarrow w0 > - w2$$
$$1 * w1 + 0 * w2 + w0 > 0 \Leftrightarrow w0 \leq - w1$$
$$1 * w1 + 1 * w2 + w0 \leq 0 \Leftrightarrow w0 \leq - w1 - w2$$

This self of inequalities is self-contradictory

⇒ Minsky & Pappert criticism of perceptron was justified in part by this result!

## Perceptrons (9.2) (cont.)

– The XOR problem can be solved using a two-layer perceptron illustrated by the following figure

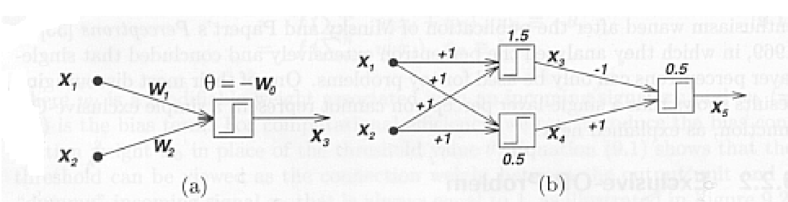



**Figure 9.4.** *Perceptrons for the two-input exclusive-OR problem: (a) the single-layer perceptron, and (b) the two-layer perceptron. Both use the step function as the activation function for each node.*

## Perceptrons (9.2) (cont.)

(x1 = 0, x2 = 0 ⇒ 0)

results at the hidden layer

0 * (+1) + 0 * (+1) = 0 < 1.5 ⇒ x3 = 0
0 * (+1) + 0 * (+1) = 0 < 0.5 ⇒ x4 = 0

results at the output layer

0 * (-1) + 0 * (+1) = 0 < 0.5 ⇒ x5 = output = 0

(x1 = 0, x2 = 1 ⇒ 1)

results at the hidden layer

0 * (+1) + 1 * (+1) = 1 < 1.5 ⇒ x3 = 0
1 * (+1) + 0 * (+1) = 1 > 0.5 ⇒ x4 = 1

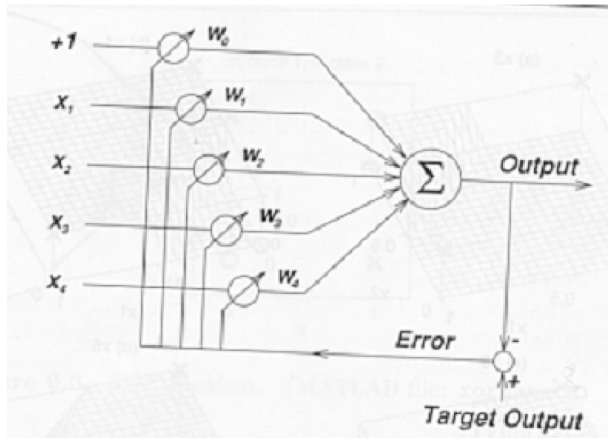
results at the output layer

0 * (-1) + 1 * (+1) = +1 > 0.5 ⇒ x5 = output = 1

In summary, multilayer perceptrons can solve nonlinearly separable problems and are thus more powerful than the single-layer perceptrons

## ADALINE (9.3)

- Developed by Widrow & Hoff, this model represents a classical example of the simplest intelligent self-learning system that can adapt itself to achieve a given modeling task




Adaline (Adaptive linear element)

## ADALINE (9.3) (cont.)

$$\textbf{output} = \sum_{i=1}^{n} w_i x_i + w_0$$

- One possible implementation of ADALINE is the following:
  - The input signals are voltages
  - The weights wi are conductances of controllable resistors
  - The network output is the summation of the currents caused by the input voltages
  - The problem consists of finding a suitable set of conductances such that the input-output behavior of ADALINE is close to a set of desired input-output data points

## ADALINE (9.3) (cont.)

- The ADALINE model can be solved using a linear least-square method, (n +1) linear parameters in order to minimize the error $\sum_{p=1}^{m}(t_p - o_p)^2$ (m training data)
- However, since this method can be slow (requires too many calculations!) if n is large, therefore Widrow & Hoff fell back on gradient descent

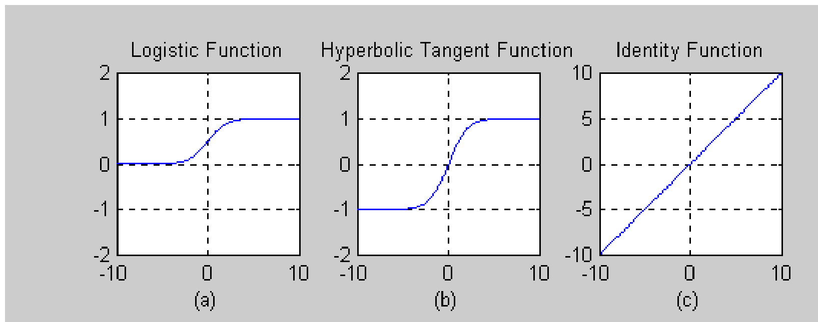
$$\text{if } E_p = (t_p - o_p)^2 \Rightarrow \frac{\partial E_p}{\partial w_i} = -2(t_p - o_p) * x_i$$

$$\text{which provides : } w^p_{next} = w^p_{now} + \eta \underbrace{(t_p - o_p) * x_i}_{g}$$

(Least-mean square (LMS) learning procedure)

## Backpropagation Multilayer Perceptrons (9.4)

- There was a change in 1985 of the reformulation of the backpropagation training method by Rumelhart
- The signum and the step functions are not differentiable, the use of logistic (hyperbolic) functions contribute for a better learning scheme
  - Logistic: f(x) = 1 / (1 + e-x)
  - Hyperbolic tangent: f(x) = tanh(x/2) = (1 – e-x) / (1 + e-x)
  - Identity: f(x) = x
- The signum function is approximated by the hyberbolic tangent function & the step function is approximated by the logistic function




Activation functions for backpropagation MLPs

## Backpropagation Multilayer Perceptrons (9.4) (cont.)

- Backpropagation learning rule

Principle:

The net input $\bar{x}$ of a node is defined as the weighted sum of the incoming signals plus a bias term:

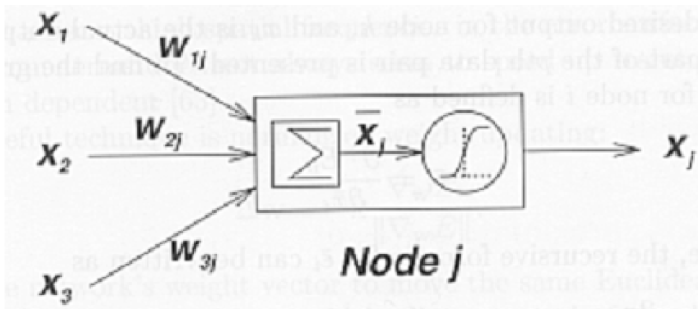
$$\bar{x}_j = \sum_i w_{ij} x_i + w_j \qquad \text{(Logistic function)}$$

$$x_j = f(\bar{x}_j) = \frac{1}{1 + \exp(-\bar{x}_j)}$$

Where: $x_i$ = ouptput of node i at any of the previous layers

$w_{ij}$ = weight associated with the link connecting nodes i & j

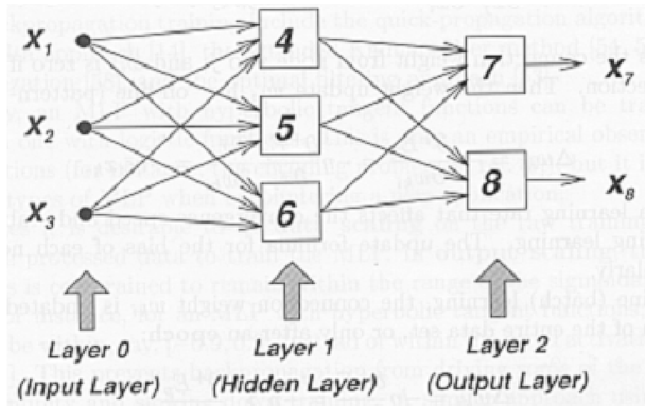
$w_j$ = bias of node j




Node j of a backpropagation MLP

## Backpropagation Multilayer Perceptrons (9.4) (cont.)

- The following figure shows a two-layer backpropagation MLP with 3 inputs in the input layer, 3 neurons in the hidden layer & 2 output neurons in the output layer




A 3-3-2 backpropagation MLP

## Backpropagation Multilayer Perceptrons (9.4) (cont.)

- The square error measure for the p-th input-output pair is defined as:

$$\mathbf{E_p} = \sum_{\mathbf{k}} (\mathbf{d_k} - \mathbf{x_k})^2$$

where: $d_k$ = desired output for node k

$x_k$ = actual output for node k when the p-th data pair is presented

- Since it is a propagation scheme, an error term $\bar{\varepsilon}_i$ for node i is needed:

$$\bar{\varepsilon}_i = \frac{\partial^+ \mathbf{E_p}}{\partial \bar{\mathbf{x}}_i}$$

Using a chain rule derivation, we obtain:

$$\mathbf{w}_{k,i}^{next} = \underbrace{\mathbf{w}_{k,i}^{now} - \eta \nabla_{w_{k,i}} \mathbf{E}}_{\text{steepest descent}} = \mathbf{w}_{k,i}^{now} - \eta \bar{\varepsilon}_i \mathbf{x}_k$$

## Backpropagation Multilayer Perceptrons (9.4) (cont.)

- Algorithm (Stochastic backpropagation)

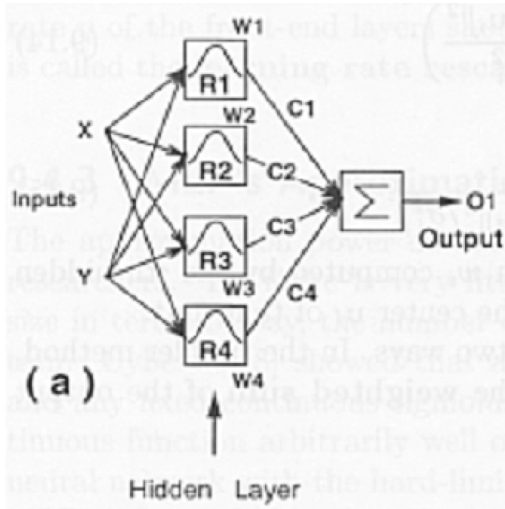
```
Begin initialize number-of-hidden-units,
w, criterion θ,η, m (training data size)
    Do m ← m + 1
       x^m ← randomly chosen pattern
               w_{k,I} ← w_{k,I} + η ε̄_i x_k
         Until ||∇E(w)|| < θ
       Return w
End.
```

## Radial Basis Function Networks (9.5)

- Architectures & Learning Methods
  - Inspired by research in regions of the cerebral cortex & the visual cortex, RBFNs have been proposed by Moody & Darken in 1988 as a supervised learning neural networks
  - The activation level of the ith receptive field unit is:

$$w_i = R_i(x) = R_i(||x - u_i|| / \sigma_i),\ i = 1, 2, \ldots, H$$

    - x is a multidimensional input vector
    - $u_i$ is a vector with same dimension as x
    - H is the number of radial basis functions called also receptive field units
    - $R_i(.)$ is the ith radial basis function with a single maximum at the origin




Single-output RBFN that uses weighted sum

## Radial Basis Function Networks (9.5) (cont.)

- Architectures & Learning Methods (cont.)
  - $R_i(.)$ is either a Gaussian function

$$R_i^G(x) = \exp\left(-\frac{\|x - u_i\|^2}{2\sigma_i^2}\right)$$

or a logistic function

$$R_i^L(x) = \frac{1}{1 + \exp[\|x - u_i\|^2 / \sigma_i^2]}$$

if $x = u_i \Rightarrow R_i^G = 1$ (Maximum) & $R_i^L = ½$ (Maximum)

## Radial Basis Function Networks (9.5) (cont.)

- Architectures & Learning Methods (cont.)
  - The output of an RBFN
    - $$d(x) = \sum_{i=1}^{i=H} c_i w_i = \sum_{i=1}^{i=H} c_i R_i(x)$$ (weighted sum)

      where $c_i$ = output value associated with the ith receptive field
    - $$d(x) = \frac{\sum_{i=1}^{i=H} c_i w_i}{\sum_{i=1}^{i=H} w_i} = \frac{\sum_{i=1}^{i=H} c_i R_i(x)}{\sum_{i=1}^{i=H} R_i(x)}$$ (weighted average)

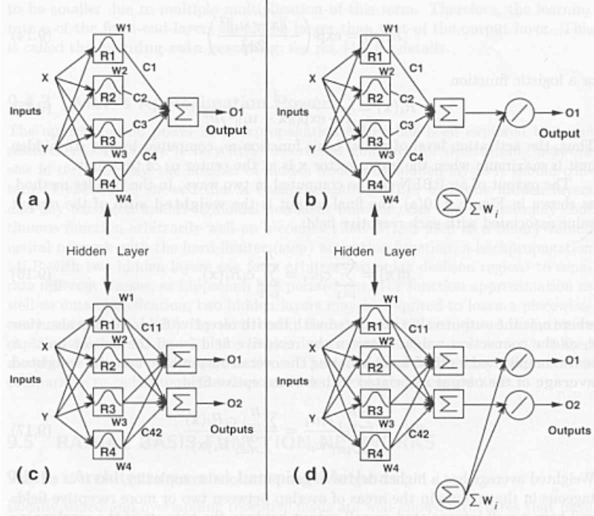



**Figure 9.10.** *Four RBFNs that possess four basis functions: (a) single-output RBFN that uses weighted sum; (b) single-output RBFN that uses weighted average; (c) two-output RBFN that uses weighted sum; (d) two-output RBFN that uses weighted average. The network in (d) is equivalent to Figure 13.3 (upper right). [Note that in (b) and (d), four connections to the lower summation unit are omitted for simplicity.]*

## Radial Basis Function Networks (9.5) (cont.)

- Architectures & Learning Methods (cont.)
  - Moody-Darken's RBFN may be extended by assigning a linear function to the output function of each receptive field

$$c_i = a_i^T x + b$$

(${a_i}$ is a parameter vector & $b_i$ is a scalar parameter)

  - Supervised adjustments of the center & shape of the receptive field (or radial basis) functions may improve RBFNs approximation capacity
  - Several learning algorithms have been proposed to identify the parameters ($u_i$, $\sigma_I$ & $c_i$) of an RBFN

## Radial Basis Function Networks (9.5) (cont.)

- Functional Equivalence to FIS

$$c_i = \vec{a}_i . \vec{x} + b_i$$

  - The extended RBFN response given by the weighted sum or the weighted average is identical to the response produced by the first-order Sugeno fuzzy inference system provided that the membership functions, the radial basis function are chosen correctly

$$d(x) = \sum_{i=1}^{i=H} \left(\vec{a}_i \vec{x} + b_i\right) w_i(x) = \sum_{i=1}^{i=H} \left(\vec{u}_i \vec{x} + v_i\right)$$

$$\text{where}: \vec{u}_i = \left[u_i^1, u_i^2, \ldots, u_i^m\right]^T, \vec{x} = \left[x_1, x_2, \ldots, x_m\right]^T$$

## Radial Basis Function Networks (9.5) (cont.)

- Functional Equivalence to FIS (CONT.)
  - While the RBFN consists of radial basis functions, the FIS comprises a certain number of membership functions
  - The FIS & the RBFN were developed on different bases, they are rooted in the same soil

## Radial Basis Function Networks (9.5)(cont.)

- Functional Equivalence to FIS (cont.)
  - Condition of equivalence between FIS & RBFN
    - RBFN & FIS use both of them the same aggregation method (weighted average & weighted sum)
    - The number of receptive field units in RBFN is equal to the number of fuzzy if-then rules in the FIS
    - Each radial basis function of the RBFN is equal to a multidimensional composite MF of the premise part of a fuzzy rule in the FIS
    - Corresponding radial basis & fuzzy rule should have the same response function

## Radial Basis Function Networks (9.5) (cont.)

- Interpolation & approximation RBFN
  - The interpolation case: each RBF is assigned to each training pattern

    Goal: Estimate a function d(.) that yields exact desired outputs for all training data

    - Our goal consists of finding $c_i$ (i = 1, 2, …, n) (n = H) such that $d(x_i) = o_i$ = desired output

      since $w_i = R_i(\|x - u_i\|) = \exp[-(x - u_i)^2 / (2\sigma_i^2)]$
      Therefore, starting with $x_i$ as centers for the RBFNs, we can write:

$$d(x) = \sum_{i=1}^{n} c_i \exp\left[-\frac{(x - x_i)^2}{2\sigma_i^2}\right]$$

## Radial Basis Function Networks (9.5) (cont.)

- Interpolation & approximation RBFN (cont.)
  - **The interpolation case (cont.)**
    - For given $\sigma_i$ (i = 1, …, n), we obtain the following n simultaneous linear equations with respect to unknown weights $c_i$ (i = 1, 2, …, n)

## Radial Basis Function Networks (9.5) (cont.)

– The interpolation case (cont.)

$$\textbf{First pattern } d(x_1) = \sum_{i=1}^{n} c_i \exp\left[-\frac{(x_1 - x_i)^2}{2\sigma_i^2}\right] = d_1$$

$$\textbf{Second pattern } d(x_2) = \sum_{i=1}^{n} c_i \exp\left[-\frac{(x_2 - x_i)^2}{2\sigma_i^2}\right] = d_2$$

$\vdots$

$$\textbf{nth pattern } d(x_n) = \sum_{i=1}^{n} c_i \exp\left[-\frac{(x_n - x_i)^2}{2\sigma_i^2}\right] = d_n$$

$$D = GC \quad \text{where } D = [d_1, d_2, \ldots, d_n]^T, C = [c_1, c_2, \ldots, c_n]^T,$$

and $G$ = exponential function values

$\Downarrow$

$C = G^{-1}D$ (optimal weights if G is nonsingular)

## Radial Basis Function Networks (9.5) (cont.)

- Interpolation & approximation RBFN (cont.)
  - **Approximation RBFN**
    - This corresponds to the case when there are fewer basis functions than there are available training patterns
    - In this case, the matrix G is rectangular & the least square methods are commonly used in order to find the vector C