Slide 0

Neural Networks

오늘의 교훈

과학에 있어서 진리란 밝혀지지않은 거짓이다

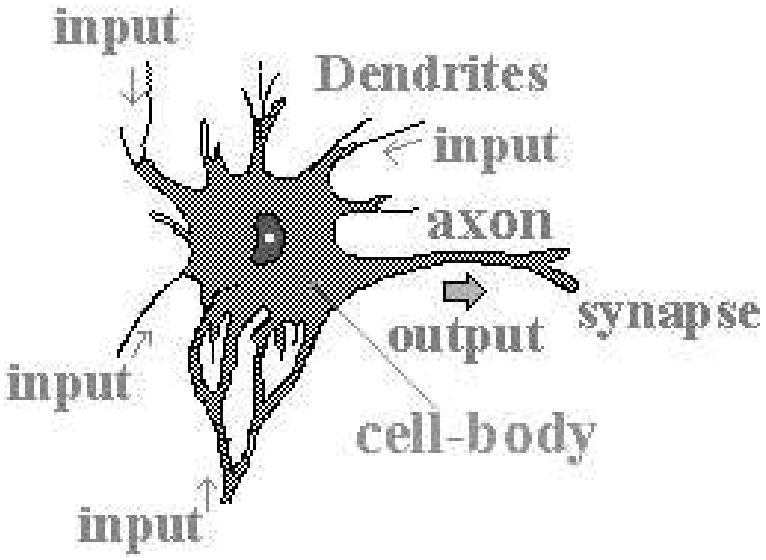
What are neural networks

Slide 1

- A biological neuron (nerve cell) has 3 parts, a cell body, an input part dendrites, and an output part axon.
- The end of axon has a set of branching fibers which are connected to dendrites of other neurons.
- Signals are sent through this route among neurons.
- These connections are called synapses.
- Neural networks are models which try to modelize the biological nerve system.
- A neural network is constructed with highly interconnected multiple neurons.
- According to the results of biological research, a neuron performs a weighted sum on the inputs and uses a nonlinear threshold value function to compute its output.

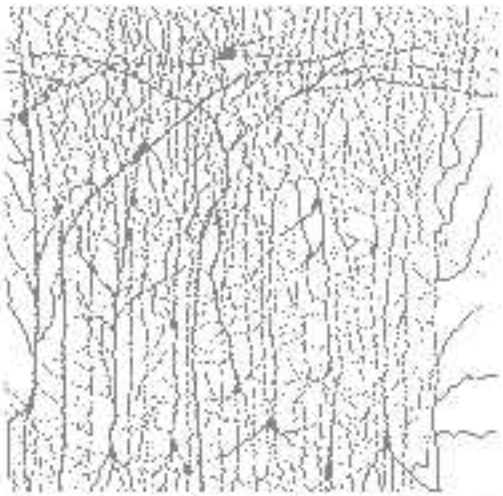
What are neural networks (계속)

Slide 2



What are neural networks (계속)

Slide 3



A typical network of neurons

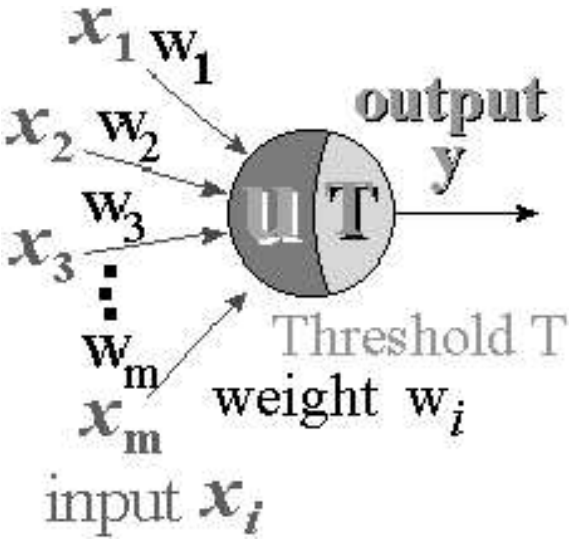
What are neural networks (계속)

Slide 4

- The biological neuron's function is modelized as the following mathematical model (artificial neural network)
- Weighted sum: Weighted sum of inputs w1x1 + w2x2 + ... + wmxm
- If the weighted sum is greater than threshold T, then the neuron will produce an output otherwise no output will be produced.
- u = w1x1 + w2x2 + ... + wmxm - T
- y = f(u)

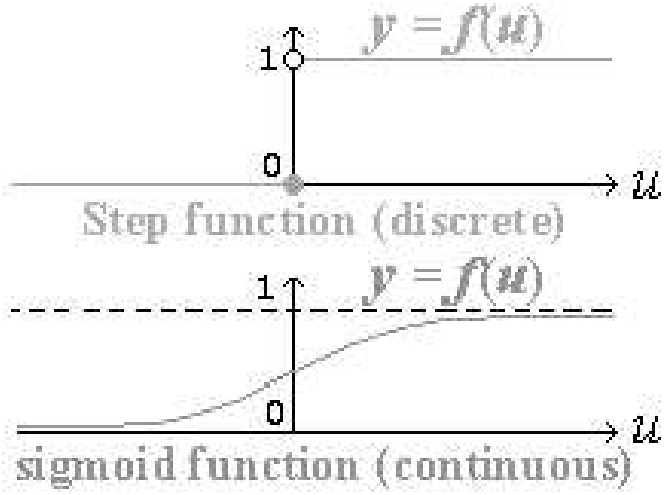
What are neural networks (계속)

Slide 5



What are neural networks (계속)

Slide 6



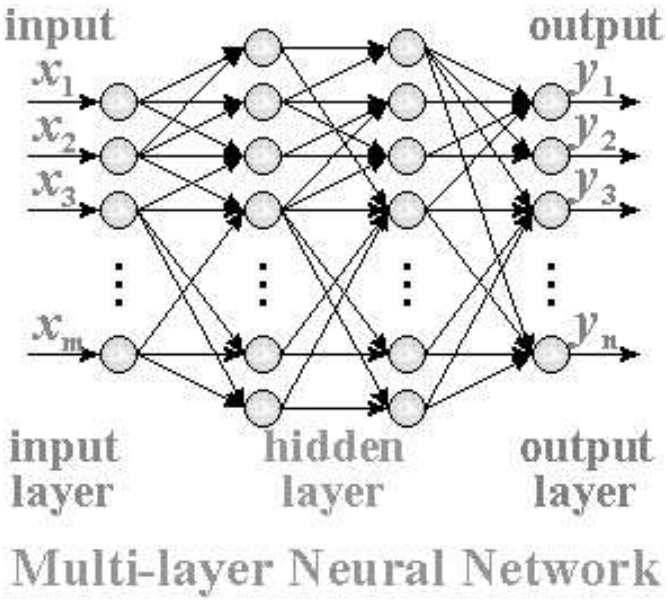
What are neural networks (계속)

Slide 7

- Based on the network's mechanism, the neural networks may be divided into the following groups:
  - (1) Multi-layered Neural Networks (Back-propagation)
  - In multi-layered neural networks, the signals flow in a single direction from input layer to output layer via hidden layer(s).

What are neural networks (계속)

Slide 8

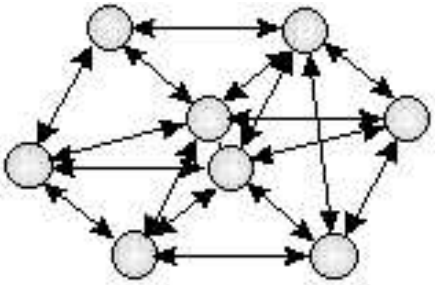


Multi-layer Neural Network

What are neural networks (계속)

Slide 9

- (2) Fully Interconnected Neural Networks (Hopfield)
- In fully interconnected neural networks, the signals travel bi-directionally among neurons.
- Signals are exchanged randomly among neurons.



Fully Interconnected One-layer Neural Network

What are neural networks (계속)

Slide 10

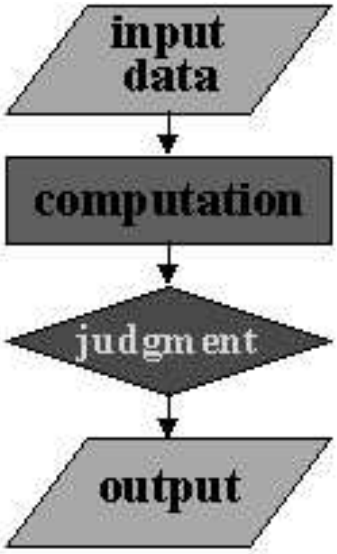
- Features of Neural Networks
  - Neural networks have several wonderful features, such as:
    * Massive Parallel Processing
      1. Traditionally, computers operate sequentially.
      2. On the other hand, humans process the information in a parallel form.
      3. Neural networks, like human, can put the parallel information processing into practice.

What are neural networks (계속)

Slide 11

What are neural networks (계속)

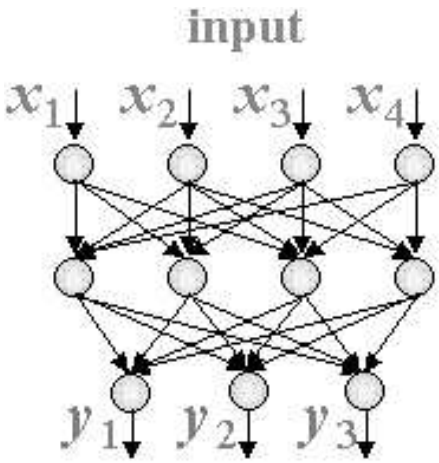
Slide 12



Serial Information Processing (Computer)

What are neural networks (계속)

Slide 13



Parallel Information Processing (Neural Network)

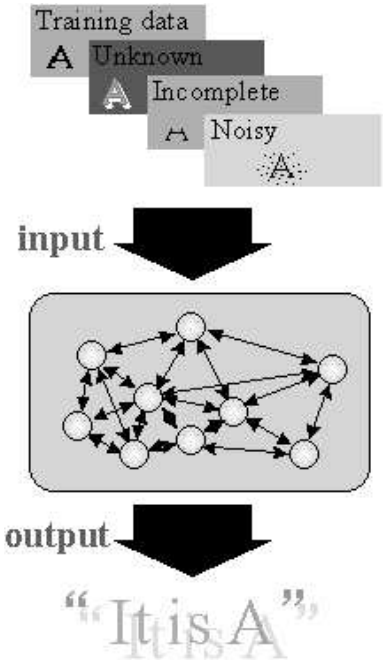
What are neural networks (계속)

Slide 14

* Learning Ability
  1. Neural networks, unlike computers, have an excellent flexibility based on learning function.
  2. They can learn from their surrounded environments and improve their performances.
  3. They can handle noisy and incomplete data.
  4. Once the neural network is trained, it can cope with noisy inputs, unknown or incomplete data.

What are neural networks (계속)

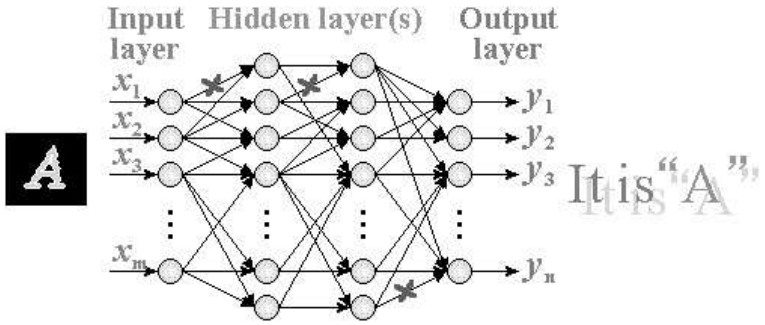
Slide 15



What are neural networks (계속)

Slide 16

* Fault Tolerance
  1. In neural networks, unlike the computers memory, information are distributed over entire neurons.
  2. Therefore, a small damage in some neurons does not stop the entire system behavior.



What are neural networks (계속)

Slide 17

- mathematical modeling technology inspired from human brain
- used for pattern recognition, prediction, and control
- history
  - in 1943, the first NN model was introduced by W. McCulloch and W. Pitts
  - in 1949, the simple learning theory by Hebb
  - in 1958, ANN "Perceptron" was introduced by F. Rosenblatt
  - in 1969, the limitation of the Perceptron was proved by M. Minsky et al.
  - 1969 $\sim$ 1980, the dark age of neural research
  - in 1980s, "Hopfield NN" by J. Hopfield and "error back-propagation" brought boom
  - in 1990s, NNs have continuously been developed and applied to various fields

McCulloch-Pitts Model

Slide 18

- the first attempt to model Neural Networks in 1943
- five assumptions
  1. the activity of a neuron is an all-or-none process
  2. a certain fixed number of synapses (> 1) must be excited within a period of latent addition for a neuron to be excited
  3. the only significant delay within the nervous system is synaptic delay
  4. the activity of any inhibitory synapse absolutely prevents excitation of the neuron at that time
  5. the structure of the interconnection network does not change with time
- assumption 1 identifies the neurons as being binary
  →they are either on or off

McCulloch-Pitts Model (계속)

Slide 19

- we can define a predicate
  * $N_i(t)$ denotes the assertion that the ith neuron fires at time t
  * $\neg N_i(t)$ denotes the assertion that the ith neuron did not fire at time t

  →with these notations, the action of certain network can be describe by propositional logic

  $N_2(t) = N_1(t-1)$ (precession)
  $N_3(t) = N_1(t-1) \vee N_2(t-1)$ (disjunction)
  $N_3(t) = N_1(t-1) \& N_2(t-1)$ (conjunction)
  $N_3(t) = N_1(t-1) \& \neg N_2(t-1)$ (conjoined negation)

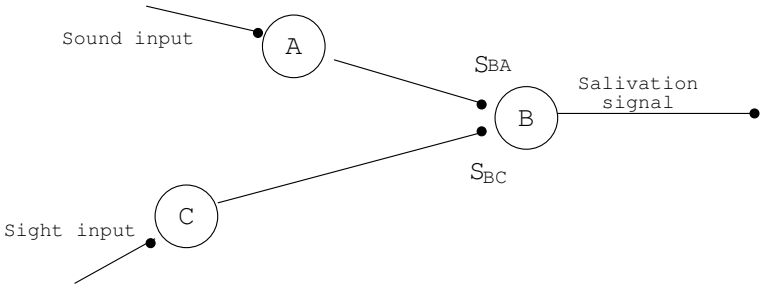
  →any network that does not have feedback connections can be described by combination of these four expressions

Hebbian Learning

- the main idea of Hebb's learning theory

> When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that $A's$ efficiency, as one of the cells firing B, is increased

Slide 20

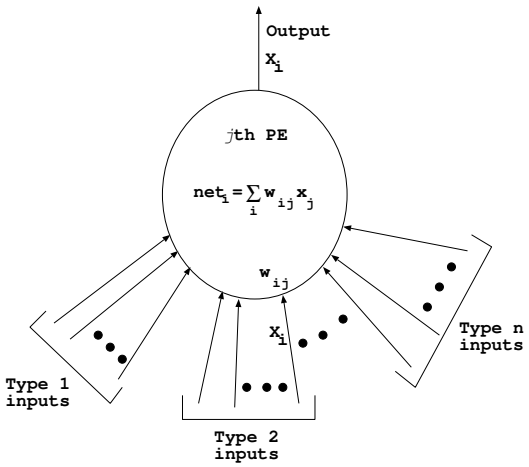


  - during learning take place, the area of the synaptic junction increased

From Neurons to ANS

- the individual computational elements of artificial neural system models are often referred to as nodes, units, or processing elements (PEs).
- a general PE model

Slide 21



From Neurons to ANS (계속)

Slide 22

- Each connection is called a weight or connection strength
  →strength of the synaptic connection between neurons
- various types
  * excitatory connections have positive weights
  * inhibitory connections have negative weights
- PE determines a net-input value: $\text{net}_i = \sum_j x_j w_{ij}$
- activation value: $a_i(t) = F_i(a_i(t-1), \text{net}_i(t))$
  →in the majority of cases, activation and net input are identical
- output function: $x_i = f_i(a_i) = f_i(\text{net}_i)$, sometimes f_i is referred to activation function

From Neurons to ANS (계속)

Slide 23

- dynamical system
  * the network can be viewed as a dynamical system
  * an example described by differential equations: $\dot{x}_i = g_i(x_i, net_i)$
  * an example
    1. $\dot{x}_i = -x_i + f_i(\text{net}_i)$
    2. for long time, the output value will reach an equilibrium, i.e., $\dot{x}_i = 0$
       $\rightarrow x_i = f_i(\text{net}_i)$
  * learning is accomplished by modification of the weight values
    $\rightarrow \dot{w}_{ij} = G_i(w_{ij}, x_i, x_j, \cdots)$
    $\rightarrow G_i$ represents the learning law
- vector formulation
  * $\text{net}_i = \sum_{j=1}^{n} x_j w_{ij}$
  * vector notation: $\text{net}_i = \mathbf{x} \cdot \mathbf{w_i} = \mathbf{x^t w_i}$

The Perceptron

- in 1958, Frank Rosenblatt invented the perceptron
- he took exception to McCulloch-Pitts model where symbolic logic was employed
- he developed a theory of statistical separability
- photoperceptron (Fig. 1.12)
  - connections have the values, +1 (excitatory), -1 (inhibitory), 0

Slide 24

  - is a learning device
    - ∗ in initial configuration, the perceptron cannot distinguish the pattern
    - ∗ through a training process, it could learn
      1. a pattern was applied to the retina (S)
      2. the stimulus was propagated through the layers until a response unit was activated
      3. if the correct response unit was active, the output of the contributing A units was increased
      4. if the incorrect R unit was active, the output of the contributing A units was decreased

The Perceptron (계속)

Slide 25

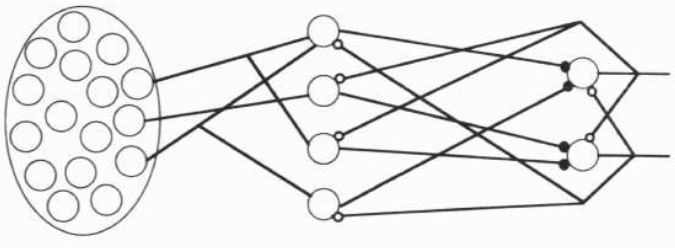


Figure 1.12 A simple photoperceptron has a sensory area, an association area, and a response area. The connections shown between units in the various areas are illustrative, and are not meant to be an exhaustive representation.

The Perceptron (계속)

Slide 26

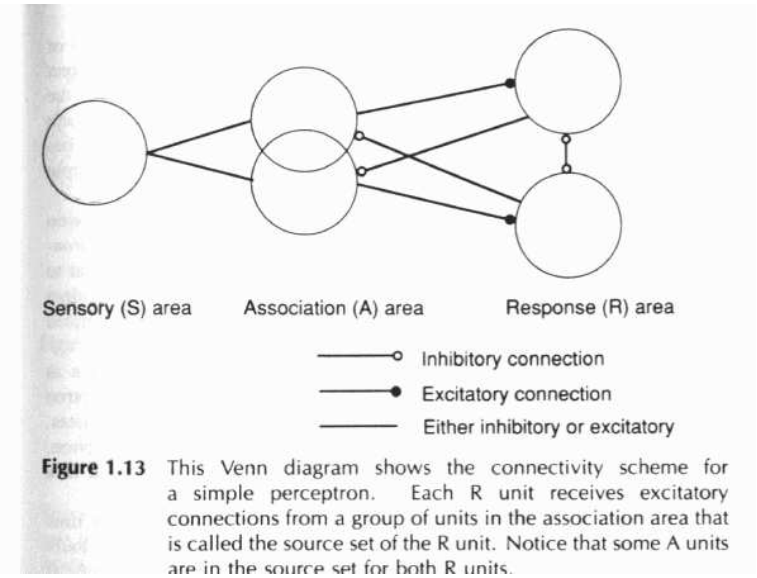


Figure 1.13 This Venn diagram shows the connectivity scheme for a simple perceptron. Each R unit receives excitatory connections from a group of units in the association area that is called the source set of the R unit. Notice that some A units are in the source set for both R units.

Learning Rules

Slide 27

- a simple mathematical model
  - $y_i(t+1) = a\left(f_i\right) \triangleq a\left(net_i\right)$

    where f_i is integration function and $a(\cdot)$ is activation function
  - integration functions
    - linear function

      $$f_i = \sum_{j=1}^{m} w_{ij} x_j - \theta_i$$

      where θ_i is threshold
    - quadratic function

      $$f_i = \sum_{j=1}^{m} w_{ij} x_j^2 - \theta_i$$

    - spherical function

      $$f_i = \rho^{-2} \sum_{j=1}^{m} (x_j - w_{ij})^2 - \theta_i$$

      where ρ and w_{ij} are the radius and the center of the sphere

Learning Rules (계속)

* polynomial function

$$f_i = \sum_{j=1}^{m}\sum_{k=1}^{m} w_{ijk}x_j x_k + x_j^{\alpha_j} + x_k^{\alpha_k} - \theta_i$$

where w_{ijk} is the weight on the conjunctive link connecting PE j and PE k to PE i and α_j and α_k are real constants

Slide 28

– activation functions

* step function

$$a(f) = \begin{cases} 1 & \text{if } f \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

* hard limiter (threshold function)

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

where sgn(·) is the signum function

Learning Rules (계속)

* ramp function

$$a(f) = \begin{cases} 1 & \text{if } f > 1 \\ f & \text{if } 0 \leq f \leq 1 \\ 0 & \text{if } f < 0 \end{cases}$$

Slide 29

* unipolar sigmoid function

$$a(f) = \frac{1}{1 + e^{-\lambda f}}$$

* bipolar sigmoid function

$$a(f) = \frac{2}{1 + e^{-\lambda f}} - 1$$

Learning Rules (계속)

Slide 30

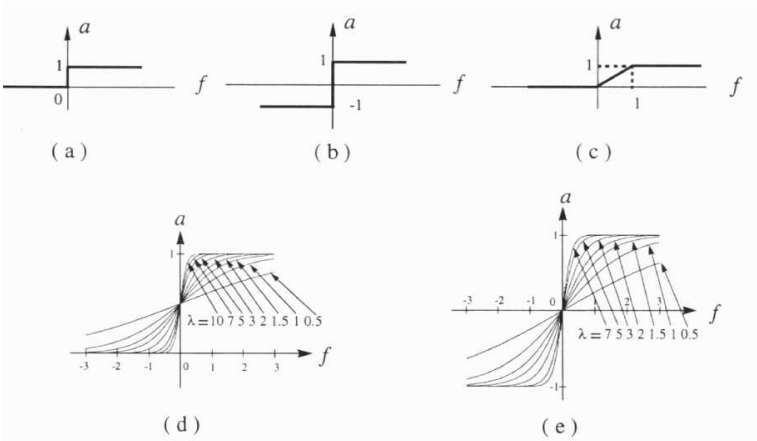


Figure 9.2 Sample activation (transfer) functions. (a) Step function. (b) Hard limiter. (c) Ramp function. (d) Unipolar sigmoid function. (e) Bipolar sigmoid function.

Learning Rules (계속)

Slide 31

- Linear Threshold Unit (LTU)
  →PE with a linear integration function and a hard limiter activation function
- Linear Graded Unit (LGU)
  →PE with a linear integration function and a graded activation function (sigmoid function)
- LTU and LGU are the most frequently used models in ANNs

Learning Rules (계속)

Slide 32

- connections
  - five basic connection geometries

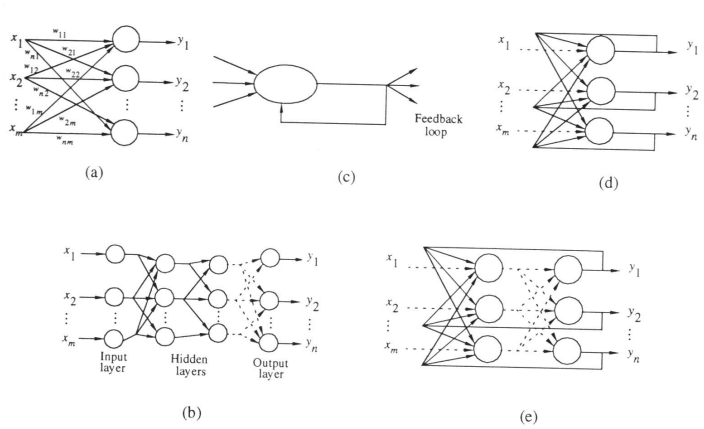


Figure 9.4 Five basic network connection geometries. (a) Single-layer feedforward network. (b) Multilayer feedforward network. (c) Single node with feedback to itself. (d) Single-layer recurrent network. (e) Multilayer recurrent network.

Learning Rules (계속)

Slide 33

- Learning Rules
  - two kinds of learning
    - parameter learning : updating of the connecting weights
    - structure learning : change in the network structure (# of PEs and connection types)
  - three categories of learning rules
    - supervised learning
    - reinforcement learning
    - unsupervised learning

Learning Rules (계속)

Slide 34

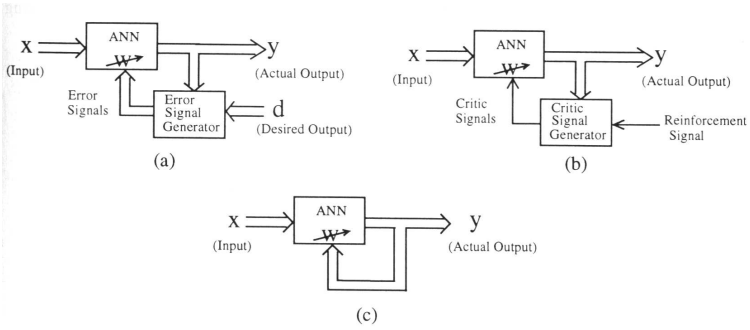


Figure 9.6 Three categories of learning. (a) Supervised learning. (b) Reinforcement learning. (c) Unsupervised learning.

Learning Rules (계속)

Slide 35

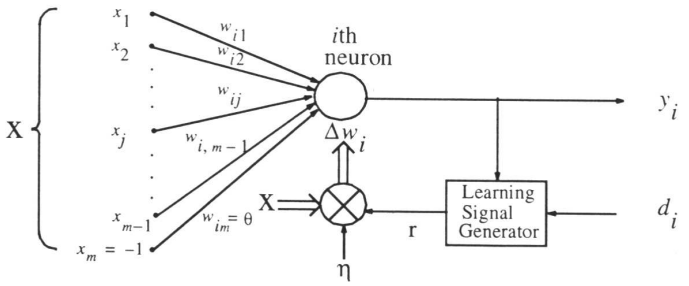


Figure 9.7 The general weight learning rule (d_i is not provided for the unsupervised learning mode).

- a general form of weight learning rule
  - $\Delta \mathbf{w}_i(t) \propto r\mathbf{x}(t)$ or $\Delta \mathbf{w}_i(t) = \eta r \mathbf{x}(t)$
    where η is a positive number called *learning constant* which determines the rate of learning r is the learning signal $r = f_r(\mathbf{w}_i, \mathbf{x}, d_i)$

Learning Rules (계속)

Slide 36

* the weight vector at learning time step $(t+1)$:
  $\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \eta f_r(\mathbf{w}_i(t), \mathbf{x}(t), d_i(t))\mathbf{x}(t)$
  ↑ is discrete-time weight modifications
* continuous-time weight modifications: $\frac{d\mathbf{w}_i(t)}{dt} = \eta r \mathbf{x}(t)$
* based on this general weight learning rules, several supervised and unsupervised weight learning rules have been developed

– Hebbian learning rule
* is an unsupervised learning rule because of no desired outputs
* $r \triangleq a(\mathbf{w}_i^T\mathbf{x}) = y_i$
* $\Delta\mathbf{w}_i = \eta a(\mathbf{w}_i^T\mathbf{x})\mathbf{x} = \eta y_i \mathbf{x}$
* if the components of the weight vector are updated:
  $\Delta w_{ij} = \eta a(\mathbf{w}_i^T\mathbf{x}) x_j = \eta y_i x_j, i = 1, 2, \ldots, n; j = 1, 2, \ldots, m$

Learning Rules (계속)

Slide 37

Example 9.2

Consider the Hebbian learning rule for an ANN with a single PE which is a LTU. There are four inputs, x_1, x_2, x_3, and x_4, to this PE. The corresponding weight vector is $\mathbf{w} = (w_1, w_2, w_3, w_4)^T$. Assume that this ANN is to be trained using the following three input vectors:

$$\mathbf{x}^{(1)} = \begin{pmatrix} 1 \\ 1.5 \\ 0.5 \\ 0 \end{pmatrix}, \quad \mathbf{x}^{(2)} = \begin{pmatrix} -0.5 \\ 1 \\ 0 \\ 1.5 \end{pmatrix}, \quad \mathbf{x}^{(3)} = \begin{pmatrix} -1 \\ 0 \\ -1 \\ -0.5 \end{pmatrix}.$$

The initial weight vector is selected as

$$\mathbf{w}^{(1)} = \begin{pmatrix} 1 \\ 0 \\ -1 \\ 0 \end{pmatrix},$$

and the learning constant is set to $\eta = 1$. Then according to the Hebbian learning rule, the following steps are taken:

Step 1: Consider the first input pattern $\mathbf{x}^{(1)}$ to update the weight vector:

$$\mathbf{w}^{(2)} = \mathbf{w}^{(1)} + \text{sgn}((\mathbf{w}^{(1)})^T\mathbf{x}^{(1)})\mathbf{x}^{(1)} = \begin{pmatrix} 1 \\ 0 \\ -1 \\ 0 \end{pmatrix} + \text{sgn}(0.5)\begin{pmatrix} 1 \\ 1.5 \\ 0.5 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \\ 1.5 \\ -0.5 \\ 0 \end{pmatrix}.$$

Learning Rules (계속)

Slide 38

Step 2: Consider the second input pattern $\mathbf{x}^{(2)}$:

$$\mathbf{w}^{(3)} = \mathbf{w}^{(2)} + \text{sgn}((\mathbf{w}^{(2)})^T\mathbf{x}^{(2)})\mathbf{x}^{(2)} = \begin{pmatrix} 2 \\ 1.5 \\ -0.5 \\ 0 \end{pmatrix} + \text{sgn}(0.5)\begin{pmatrix} -0.5 \\ 1 \\ 0 \\ 1.5 \end{pmatrix} = \begin{pmatrix} 1.5 \\ 2.5 \\ -0.5 \\ 1.5 \end{pmatrix}.$$

Step 3: Consider the third input pattern $\mathbf{x}^{(3)}$:

$$\mathbf{w}^{(4)} = \mathbf{w}^{(3)} + \text{sgn}((\mathbf{w}^{(3)})^T\mathbf{x}^{(3)})\mathbf{x}^{(3)} = \begin{pmatrix} 1.5 \\ 2.5 \\ -0.5 \\ 1.5 \end{pmatrix} + \text{sgn}(-1.75)\begin{pmatrix} -1 \\ 0 \\ -1 \\ -0.5 \end{pmatrix} = \begin{pmatrix} 2.5 \\ 2.5 \\ 0.5 \\ 2 \end{pmatrix}.$$

It can be verified that $\text{sgn}((\mathbf{w}^{(4)})^T\mathbf{x}^{(1)}) = \text{sgn}((\mathbf{w}^{(4)})^T\mathbf{x}^{(2)}) = 1$ and $\text{sgn}((\mathbf{w}^{(4)})^T\mathbf{x}^{(3)}) = -1$. This means that the inputs ($\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$) that caused the PE to fire previously will cause it to fire again in the future with the final learned weights. This is also true for the input $\mathbf{x}^{(3)}$ which inhibits the firing of the PE. Thus, the final weights have captured the coincidence relationship of input-output training pairs.

Learning Rules (계속)

- single-layer perceptron
  - learning process

$$y_i^{(k)} = a(\mathbf{w}_i^T\mathbf{x}^{(k)}) = a\left(\sum_{j=1}^{m} w_{ij}x_j^{(k)}\right) = d_i^{(k)}, i = 1, 2, \cdots, n; k = 1, 2, \cdots, p$$

    where $y_i^{(k)}$ is actual output pattern and $d_i^{(k)}$ is target pattern

Slide 39

  - learning rule (simple perceptrons with LTUs)

    $y_i(k) = sgn(\mathbf{w}_i^T\mathbf{x}^{(k)}) = d_i^{(k)}, i = 1, 2, \cdots, n; k = 1, 2, \cdots, p$

    $\mathbf{w}_i^T\mathbf{x}(k) = 0$ is the hyperplane (called *decision plane*)
  - ex. 10.1)
    1. classify the six patterns in a two-dimensional space

       $\{[-1, 0]^T, [-1.5, -1]^T, [-1, -2]^T\}$: class 1

       $\{[2, 0]^T, [2.5, -1]^T, [1, -2]^T\}$: class 2
    2. $g(x) = -2x_1 + x_2 + 2 = 0$ (fig 10.2 (a))
    3. $y^{(k)} = sgn(w_1x_1^{(k)} + w_2x_2^{(k)} - \theta)$

       $\rightarrow w_1 = -2, w_2 = 1, \theta = -2$ (fig 10.2 (c))

Learning Rules (계속)

Slide 40

4. $y^{(k)} = sgn(\mathbf{w}^T\mathbf{x}^{(k)}) = d^{(k)}$

where

$$x^{(1)} = \begin{pmatrix} -1 \\ 0 \\ -1 \end{pmatrix}, x^{(2)} = \begin{pmatrix} -1.5 \\ -1 \\ -1 \end{pmatrix}, x^{(3)} = \begin{pmatrix} -1 \\ -2 \\ -1 \end{pmatrix},$$

$$x^{(4)} = \begin{pmatrix} 2 \\ 0 \\ -1 \end{pmatrix}, x^{(5)} = \begin{pmatrix} 2.5 \\ -1 \\ -1 \end{pmatrix}, x^{(6)} = \begin{pmatrix} 1 \\ -2 \\ -1 \end{pmatrix},$$

$d^{(1)} = d^{(2)} = d^{(3)} = +1, d^{(4)} = d^{(5)} = d^{(6)} = -1, \mathbf{w} = [w_1, w_2, w_3]^T$

Learning Rules (계속)

Slide 41

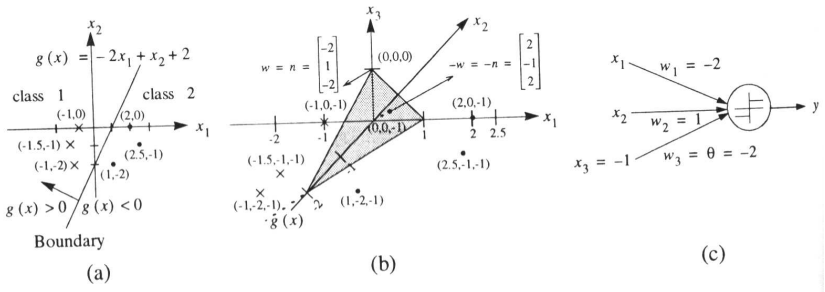


Figure 10.2 Classification problem using single-LTU perception in Example 10.1.

Learning Rules (계속)

Slide 42

* learning rule

1. $r \triangleq d_i - y_i$, where $y_i = sgn(\mathbf{w}_i^T \mathbf{x})$ and d_i takes the values of ± 1
2. $$\Delta w_{ij} = \eta[d_i - sgn(\mathbf{w}_i^T \mathbf{x})]x_j = \begin{cases} 2\eta d_i x_j & \text{if } y_i \neq d_i \\ 0 & \text{otherwise} \end{cases}$$
3. ex. 10.2 ($\eta = 0.5$)

Learning Rules (계속)

Slide 43

Example 10.2

Let us consider another simple classification problem where the input space is a one-dimensional space, that is, a real line:

Class 1: $x^{(1)} = 0.5, \quad x^{(3)} = 2; \quad d^{(1)} = d^{(3)} = +1.$

Class 2: $x^{(2)} = -1, \quad x^{(4)} = -2; \quad d^{(2)} = d^{(4)} = -1.$

The simple perceptron with a single LTU for solving this problem is shown in Figure 10.5(a). The unknown weights w_1 and w_2 in this figure are trained using the following augmented input vectors:

$$\mathbf{x}^{(1)} = \begin{pmatrix} x^{(1)} \\ -1 \end{pmatrix} = \begin{pmatrix} 0.5 \\ -1 \end{pmatrix}, \quad \mathbf{x}^{(2)} = \begin{pmatrix} -1 \\ -1 \end{pmatrix}, \quad \mathbf{x}^{(3)} = \begin{pmatrix} 2 \\ -1 \end{pmatrix}, \quad \mathbf{x}^{(4)} = \begin{pmatrix} -2 \\ -1 \end{pmatrix},$$

which will be "recycled" for training if necessary. For convenience, let us arbitrarily choose $\eta = 0.5$ and an initial weight $\mathbf{w}^{(1)} = [-2, 1.5]^T$. Then the training proceeds as follows.

Step 1: When $\mathbf{x}^{(1)}$ is the input, the corresponding actual output and weights are, respectively,

$$y^{(1)} = \text{sgn}\left([-2, 1.5] \begin{pmatrix} 0.5 \\ -1 \end{pmatrix} \right) = -1 \neq d^{(1)},$$

$$\mathbf{w}^{(2)} = \mathbf{w}^{(1)} + \mathbf{x}^{(1)} = \begin{pmatrix} -1.5 \\ 0.5 \end{pmatrix}.$$

Learning Rules (계속)

Slide 44

Step 2: When $\mathbf{x}^{(2)}$ is the input, the corresponding actual output and weights are, respectively,

$$y^{(2)} = \text{sgn}\left([-1.5, 0.5] \begin{pmatrix} -1 \\ -1 \end{pmatrix} \right) = 1 \neq d^{(2)},$$

$$\mathbf{w}^{(3)} = \mathbf{w}^{(2)} - \mathbf{x}^{(2)} = \begin{pmatrix} -0.5 \\ 1.5 \end{pmatrix}.$$

Step 3: Similarly, when $\mathbf{x}^{(3)}$ is the input, the corresponding actual output and weights are, respectively,

$$y^{(3)} = \text{sgn}\left([-0.5, 1.5] \begin{pmatrix} 2 \\ -1 \end{pmatrix} \right) = -1 \neq d^{(3)},$$

$$\mathbf{w}^{(4)} = \mathbf{w}^{(3)} + \mathbf{x}^{(3)} = \begin{pmatrix} 1.5 \\ 0.5 \end{pmatrix}.$$

Step 4: Finally, when $\mathbf{x}^{(4)}$ is the input, the corresponding actual output and weights are, respectively,

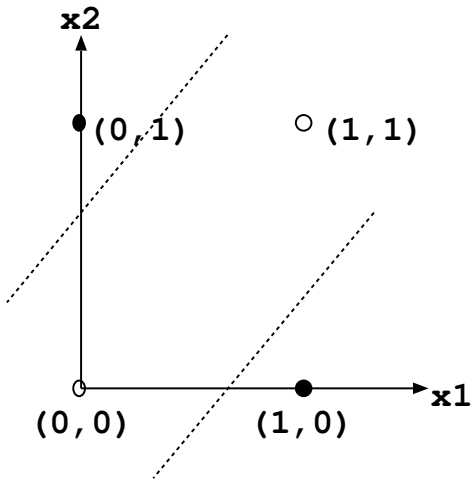
$$y^{(4)} = \text{sgn}\left([1.5, 0.5] \begin{pmatrix} -2 \\ -1 \end{pmatrix} \right) = -1 \neq d^{(4)},$$

$$\mathbf{w}^{(5)} = \mathbf{w}^{(4)}.$$

Learning Rules (계속)

Slide 45

* limitations : linearly separable (no decision-plane for XOR problem)



Adaline

- in 1962, Widrow introduces *Adaline* (*Ada*ptive *Lin*ear *E*lement)
- PE with a linear activation function
  - a supervised learning problem
  - training patterns, $\{(\mathbf{x}^{(1)}, d^{(1)}), (\mathbf{x}^{(2)}, d^{(2)}), \cdots, (\mathbf{x}^{(p)}, d^{(p)})\}$
  - the goal is to find a correct set of weight w_i

Slide 46

$$\sum_{j=1}^{m} w_j x_j^{(k)} = d^{(k)}, \qquad k = 1, 2, \cdots, p$$

  - the solutions for $\mathbf{w}$ exist if the patterns are linearly independent
  - cost function $E(\mathbf{w})$: measures of system performance error

$$\begin{aligned} E(\mathbf{w}) &= \frac{1}{2}\sum_{k=1}^{p}(d^{(k)} - y^{(k)})^2 \\ &= \frac{1}{2}\sum_{k=1}^{p}(d^{(k)} - \mathbf{w}^T\mathbf{x}^{(k)})^2 \end{aligned}$$

Adaline (계속)

$$= \frac{1}{2}\sum_{k=1}^{p}(d^{(k)} - \sum_{j=1}^{m} w_j x_j^{(k)})^2$$

    * find the weights to minimize $E(\mathbf{w})$
  - gradient-descent algorithm

Slide 47

    * $\Delta\mathbf{w} = \eta\nabla_{\mathbf{w}} E(\mathbf{w})$

$$\Delta w_j = \eta\frac{\partial E}{\partial w_j} = \eta\sum_{k=1}^{p}(d^{(k)} - \mathbf{w}^T\mathbf{x}^{(k)})x_j^{(k)}, j = 1, 2, \cdots, m$$

    * if the change are made for individual patterns
      $\Delta w_j = \eta(d^{(k)} - \mathbf{w}^T\mathbf{x}^{(k)})x_j^{(k)}$
      →called *Adaline learning rule*, *Widrow-Hoff learning rule*, *least mean square (LMS) rule*
    * *a linear network* : an array of Adalines

Linear Graded unit

Slide 48

- activation function is nonlinear and differentiable
- cost function

$$\begin{aligned} E(\mathbf{w}) &= \frac{1}{2}\sum_{k=1}^{p}\sum_{i=1}^{n}(d_i^{(k)} - y_i^{(k)})^2 \\ &= \frac{1}{2}\sum_{k=1}^{p}\sum_{i=1}^{n}(d_i^{(k)} - a(\mathbf{w}_i^T\mathbf{x}^{(k)}))^2 \\ &= \frac{1}{2}\sum_{k=1}^{p}\sum_{i=1}^{n}(d_i^{(k)} - a(\sum_{j=1}^{m} w_{ij}x_j^{(k)}))^2 \end{aligned}$$

  - applying the gradient-descent algorithm

$$\frac{\partial E}{\partial w_{ij}} = -\sum_{k=1}^{p}[d_i^{(k)} - a(net_i^{(k)})]a'(net_i^{(k)})x_j^{(k)}$$

  where $net_i^{(k)} \triangleq \mathbf{w}_i^T\mathbf{x}^{(k)}$ and $a'(net_i^{(k)}) = \frac{\partial a(net_i^{(k)})}{\partial net_i^{(k)}}$

Linear Graded unit (계속)

Slide 49

- if the change are made for individual patterns and the learning constant is η
  $\Delta w_{ij} = -\eta\frac{\partial E}{\partial w_{ij}} = \eta[d_i^{(k)} - a(net_i^{(k)})]a'(net_i^{(k)})x_j^{(k)}$
- called *delta learning rule*
- $r \triangleq [d_i - a(\mathbf{w}_i^T\mathbf{x})]a'(\mathbf{w}_i^T\mathbf{x})$
- in a nonlinear activation function, the learning process become stuck at a local minimum
- advantages of nonlinear activation functions
  - keep the outputs between bounds (± 1) $\rightarrow$arbitrary number of layers feasible
  - make possible the solutions of problems that are not possible with linear units
  - map nonlinear patterns
  - multilayer nonlinear feedforward network does not have linearly independent restriction

Multilayer feedforward networks

Slide 50

- LTU and LGU has limitations
  - LTU limitation: linearly separable
  - LGU limitation: linearly independent
- multilayer networks overcome these limitations
  - ex. 10.4

Multilayer feedforward networks (계속)

Slide 51

Example 10.4

This example illustrates how a linearly nonseparable problem is transformed to a linearly separable problem by a space transformation and thus can be solved by a multilayer perceptron network with LTUs. The problem we consider is the XOR problem. As shown in Fig. 10.6(a), the input patterns and the corresponding desired outputs are

$$\left(\mathbf{x}^{(1)} = \begin{pmatrix}0\\0\end{pmatrix}, d^{(1)} = 1\right); \qquad \left(\mathbf{x}^{(2)} = \begin{pmatrix}0\\1\end{pmatrix}, d^{(2)} = -1\right);$$

$$\left(\mathbf{x}^{(3)} = \begin{pmatrix}1\\0\end{pmatrix}, d^{(3)} = -1\right); \qquad \left(\mathbf{x}^{(4)} = \begin{pmatrix}1\\1\end{pmatrix}, d^{(4)} = 1\right).$$

Obviously, the four input patterns are not linearly separable in the input space (or the *pattern space*). We arbitrarily use two lines to partition the input space into three subspaces [see Fig.

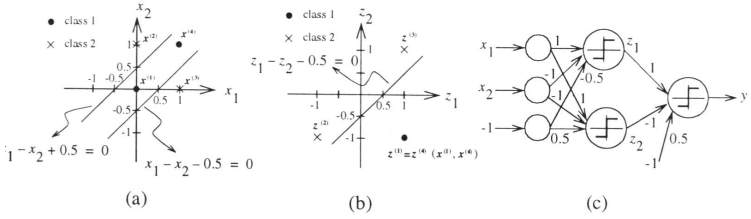


Figure 10.6 Multilayer perceptron for the XOR problem in Example 10.4. (a) Input (pattern) space (linearly nonseparable). (b) Image space (linearly separable). (c) A multilayer perceptron network for the XOR problem.

Multilayer feedforward networks (계속)

Slide 52

10.6(a)] in such a way that each subspace contains patterns belonging to the same class. The two selected lines are

$$x_1 - x_2 + 0.5 = 0 \qquad \text{and} \qquad x_1 - x_2 - 0.5 = 0.$$

Two LTUs are used to perform the partitioning created by these two selected lines. Hence, their respective outputs are

$$z_1 = \text{sgn}\,(x_1 - x_2 + 0.5) \qquad \text{and} \qquad z_2 = \text{sgn}\,(x_1 - x_2 - 0.5)\,.$$

These two LTUs are put in the hidden layer of the final network in Fig. 10.6(c). Looking at the z_1-z_2 space, also called the *image space,* the original training patterns become

$$\left(\mathbf{z}^{(1)} = \begin{pmatrix}1\\-1\end{pmatrix}, d^{(1)} = 1\right); \qquad \left(\mathbf{z}^{(2)} = \begin{pmatrix}-1\\-1\end{pmatrix}, d^{(2)} = -1\right);$$

$$\left(\mathbf{z}^{(3)} = \begin{pmatrix}1\\1\end{pmatrix}, d^{(3)} = -1\right); \qquad \left(\mathbf{z}^{(4)} = \begin{pmatrix}1\\-1\end{pmatrix}, d^{(4)} = 1\right).$$

The transformed training patterns in the image space are shown in Fig. 10.6(b). It is observed that the original input patterns $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(4)}$ coincide in the image space and that the transformed training patterns also become linearly separable in the image space. The remaining classification problem becomes what we have dealt with previously. A boundary can be chosen arbitrarily, say $z_1 - z_2 - 0.5 = 0$ [see Fig. 10.6(b)], to separate these patterns. The function of the selected line is performed by the third LTU in the output layer, whose output is

$$y = \text{sgn}\,(z_1 - z_2 - 0.5)\,.$$

The final network for solving the XOR problem is shown in Fig. 10.6(c).

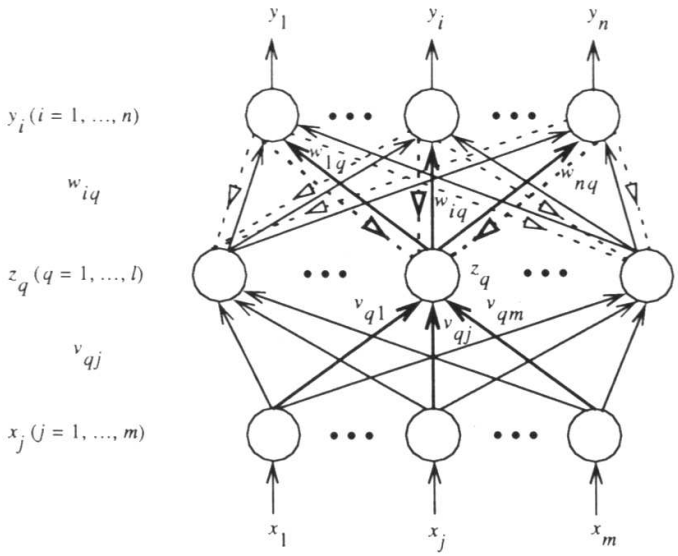
Back Propagation Learning Algorithm

Slide 53

- one of the most important development in neural networks [Werbos, 1974; LeCun, 1985; Parker, 1985; Rumelhart et al 1986]
- *back-propagation networks*: multilayer feedforward network with the BP learning algorithm
- learning rule
  - gradient-descent method
  - two phases (given input-output pair $(\mathbf{x}^{(k)}, \mathbf{d}^{(k)})$)
    1. input pattern $\mathbf{x}^{(k)}$ is propagated to produce actual output $\mathbf{y}^{(k)}$
    2. error signals $\mathbf{d}^{(k)} - \mathbf{y}^{(k)}$ are back-propagated from output layer to the previous layers to update weights
  - three-layer back-propagation network

Back Propagation Learning Algorithm (계속)

Slide 54



Back Propagation Learning Algorithm (계속)

Slide 55

* $net_q = \sum_{j=1}^{m} v_{qj} x_j$
* $z_q = a(net_q) = a(\sum_{j=1}^{m} v_{qj} x_j)$
* $net_i = \sum_{q=1}^{l} w_{iq} z_q = \sum_{q=1}^{l} w_{iq} a(\sum_{j=1}^{m} v_{qj} x_j)$
* $y_i = a(net_i) = a(\sum_{q=1}^{l} w_{iq} z_q) = a(\sum_{q=1}^{l} w_{iq} a(\sum_{j=1}^{m} v_{qj} x_j))$
* cost function
  $E(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{n} (d_i - y_i)^2 = \frac{1}{2} \sum_{i=1}^{n} [d_i - a(net_i)]^2 =$
  $\frac{1}{2} \sum_{i=1}^{n} [d_i - a(\sum_{q=1}^{l} w_{iq} z_q)]^2$
* gradient-descent method (in hidden-to-output connections)
  $\Delta w_{iq} = -\eta \frac{\partial E}{\partial w_{iq}}$
  $\Delta w_{iq} = -\eta [\frac{\partial E}{\partial y_i}][\frac{\partial y_i}{\partial net_i}][\frac{\partial net_i}{\partial w_{iq}}] = \eta [d_i - y_i][a'(net_i)][z_q] \triangleq \eta \delta_{oi} z_q$
  where $\delta_{oi} \triangleq -\frac{\partial E}{\partial net_i} = [d_i - y_i][a'(net_i)]$ is the error signal at output node i

$$\Delta v_{qj} = -\eta [\frac{\partial E}{\partial v_{qj}}] = -\eta [\frac{\partial E}{\partial net_q}][\frac{\partial net_q}{\partial v_{qj}}] = -\eta [\frac{\partial E}{\partial z_q}][\frac{\partial z_q}{\partial net_q}][\frac{\partial net_q}{\partial v_{qj}}]$$

Back Propagation Learning Algorithm (계속)

$$= \eta \sum_{i=1}^{n} [(d_i - y_i) a^{'}(net_i) w_{iq}][a^{'}(net_q)][x_j]$$

$$= \eta \sum_{i=1}^{n} [\delta_{oi} w_{iq}] a^{'}(net_q) x_j = \eta \delta_{hq} x_j$$

Slide 56

where $\delta_{hq} = -[\frac{\partial E}{\partial z_q}][\frac{\partial z_q}{\partial net_q}] = a^{'}(net_q) \sum_{i=1}^{n} \delta_{oi} w_{iq}$ are error signal at hidden node q

- ☛ called *generalized delta learning rule*
- ☛ error signals δ_{oi} are propagating backward
- ☛ can be extended to the network with more than one hidden layer by chain rule
  $\rightarrow \Delta w_{ij} = \eta \delta_i x_j = \eta \delta_{output-i} \cdot x_{input-j}$
- ☛ bipolar sigmoid function is used as the activation function
  $\rightarrow \delta_{oi} = \frac{1}{2}(1 - y_i^2)[d_i - y_i]$
  $\rightarrow \delta_{hq} = \frac{1}{2}(1 - z_q^2) \sum_{i=1}^{n} \delta_{oi} w_{iq}$

Back Propagation Learning Algorithm (계속)

Slide 57

- Learning Algorithm
  - Let
    - Q feedforward layers, $q = 1, 2, \cdots, Q$
    - ${}^{q}net_i$ and ${}^{q}y_i$ denotes the input and output of the ith unit in the q layer
    - m input nodes and n output nodes
    - ${}^{q}w_{ij}$ denote the weight from ${}^{q-1}y_j$ to ${}^{q}y_i$
    - training pairs $\{(\mathbf{x}^{(k)}, \mathbf{d}^{(k)}) \mid k = 1, 2, \cdots, p\}$, $x_{m+1}^{(k)} = -1$

step 0 initialization

- choose $\eta > 0$ and E_{max} (maximum tolerable error)
- initialize the weights to small random values
- set $E = 0$ and $k = 1$

step 1 training loop

- apply kth input pattern to the input layer ($q = 1$): ${}^{q}y_i = {}^{1}y_i = x_i^{(k)}$ for all i

Back Propagation Learning Algorithm (계속)

Slide 58

step 2 forward propagation

* propagate the signal forward through the network using
  ${}^{q}y_i = a({}^{q}net_i) = a(\sum_j {}^{q}w_{ij}\ {}^{q-1}y_j)$

step 3 output error measure

* compute the error value for the output layer
  $E = \frac{1}{2}\sum_{i=1}^{n}(d_i^{(k)} - {}^{Q}y_i)^2 + E, {}^{Q}\delta_i = (d_i^{(k)} - {}^{Q}y_i)a'({}^{Q}net_i)$

step 4 error back-propagation

* propagate the errors backward to update the weights
  $\Delta\ {}^{q}w_{ij} = \eta\ {}^{q}\delta_i\ {}^{q-1}y_j$ and ${}^{q}w_{ij}^{new} = {}^{q}w_{ij}^{old} + \Delta\ {}^{q}w_{ij}$
  ${}^{q-1}\delta_i = a'({}^{q-1}net_i)\sum_j {}^{q}w_{ji}\ {}^{q}\delta_j$ for $q = Q, Q-1, \cdots, 2$.

step 5 one epoch looping

* check all training data has been cycled, if not, go to step 1; otherwise, go to step 6

step 6 total error checking

* if $E < E_{max}$, then terminate; otherwise $E = 0, k = 1$, and go to step 1

Back Propagation Learning Algorithm (계속)

Slide 59

- learning step and epoch
  - a learning step indicates learning of a single training pattern
  - one epoch indicates learning of all training patterns
- universal approximators
  - three layer network with squashing activation function and linear or polynomial integration functions can approximate any function with desired degree of accuracy if sufficiently many hidden units are available
  - squashing function
    * $a : R \rightarrow [0, 1]$ (or $[-1, 1]$)
    * it is nondecreasing and $\lim_{\lambda\rightarrow\infty} a(\lambda) = 1$, and $\lim_{\lambda\rightarrow-\infty} a(\lambda) = 0$ (or -1)
  - four layers NN with small hidden units is better than three layers NN with many hidden units
  - BP learning algorithms can make a neural network stuck at the local minima

Back Propagation Learning Algorithm (계속)

Slide 60

- Learning factors of BP
  - initial weights
    - initialize at small random values
    - equal initial weights cannot train the network properly
    - one way to choose the weight w_{ij} in the range of $[-3/\sqrt{k_i}, 3/\sqrt{k_i}$ where k_i is the number of input links of PE i
  - learning constant
    - a large η could speed up the convergence but might result in overshooting
    - a small η has a complementary effect
    - values of η ranging from 10^{-3} to 10
  - cost function
    - other cost function can be used
    - ex.) $E = \frac{1}{p}\sum_i (d_i - y_i)^p$, where $1 \leq p < \infty$

Back Propagation Learning Algorithm (계속)

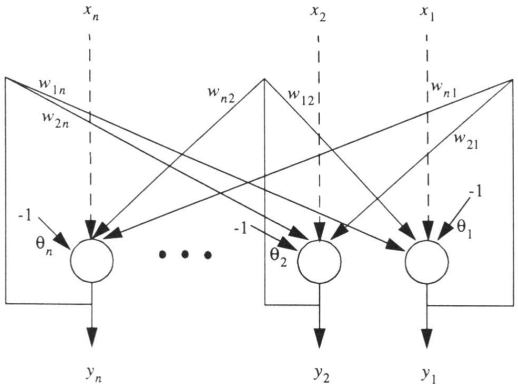
Slide 61

  - momentum
    - if learning constant η is small, then learning speed can be very slow
    - if learning constant η is large, then learning can oscillate
    - large learning constant without divergent oscillations
      - addition of a momentum term to the normal gradient-descent method
      - $\Delta w(t) = -\eta \nabla E(t) + \alpha \Delta w(t-1)$ where $\alpha \in [0, 1]$ is a momentum parameter
  - update rules
    - can use other methods not gradient-descent (or steepest-descent)
  - training data and generations
    - training data should cover the entire expected input space
    - during the training process, select data randomly
    - overfitting
      - if the change of input is small, then cost function should not change
  - number of hidden nodes
    - difficult to solve because the complexity and nonlinearity properties of NN

Single-layer feedback networks and associative memories

Slide 62

- Hopfield networks [1982, 1984]
  - single-layer feedback network (or recurrent network) with symmetric weights
  - two application areas
    - associative memory
    - optimization problems



Single-layer feedback networks and associative memories (계속)

Slide 63

- discrete Hopfield networks
  - operations
    - an input pattern is first applied
    - the network's output is initialized
    - input pattern is removed
    - the initialized output becomes new, updated inputs through the feedback connections
    - continues this process until the network reached its equilibrium

Single-layer feedback networks and associative memories (계속)

Slide 64

- properties
  - no self-feedback $\rightarrow w_{ii} = 0$ and w_{ij} for $i = 1, 2, \cdots, n, i \neq j$
  - weights are symmetric $\rightarrow w_{ij} = w_{ji}$
  - the evolving rule (or update rule)

$$y_i(k+1) = sgn\left(\sum_{j=1(j\neq i)}^{n} w_{ij} y_j^{(k)} + x_i - \theta_i \right), i = 1, 2, \cdots, n$$

- two update rules
  - asynchronous fashion: for a given time only a single node is updated and the output is used for updating next randomly chosen node
  - synchronous fashion: for a given time all node are updated

Single-layer feedback networks and associative memories (계속)

Slide 65

- ex. 11.1)
  - $w_{12} = w_{21} = -1, w_{11} = w_{22} = 0, x_1 = x_2 = 0$, and $\theta_1\ \theta_2 = 0$, set the $y^{(0)} = [-1, -1]^T$
  - asynchronous update
    $y_1^{(1)} = sgn(w_{12} y_2^{(0)}) = sgn[(-1)(-1)] = 1$
    $y_2^{(2)} = sgn(w_{21} y_1^{(1)}) = sgn[(-1)(1)] = -1$
  - no further update $\rightarrow$stable state
  - synchronous update

$$\mathbf{y}^{(1)} = \begin{pmatrix} sgn[w_{12} y_2^{(0)}] \\ sgn[w_{21} y_1^{(0)}] \end{pmatrix} = \begin{pmatrix} sgn[(-1)(-1)] \\ sgn[(-1)(-1)] \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$\mathbf{y}^{(2)} = \begin{pmatrix} sgn[w_{12} y_2^{(1)}] \\ sgn[w_{21} y_1^{(1)}] \end{pmatrix} = \begin{pmatrix} sgn[(-1)(1)] \\ sgn[(-1)(1)] \end{pmatrix} = \begin{pmatrix} -1 \\ -1 \end{pmatrix}$$

  - a cycle of two states $\rightarrow$no equilibrium state

Single-layer feedback networks and associative memories (계속)

- stability property
  - energy function E

$$E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1(j\neq i)}^{n} w_{ij}y_i y_j - \sum_{i=1}^{n} x_i y_i + \sum_{i=1}^{n}\theta_i y_i$$

  - the E will decrease if the network is stable

Slide 66

  - proof
    - $\Delta E = E(y_i^{(k+1)}) - E(y_i^{(k)}) =$
      $-\left(\sum_{j=1(j\neq i)}^{n} w_{ij}y_j^{(k)} + x_i - \theta_i\right)(y_i^{(k+1)} - y_i^{(k)}) = -(net_i)\Delta y_i$
    - $y_i^{(k)} = +1, y_i^{(k+1)} = -1$ case
      $\rightarrow \Delta y_i = -2$ and $net_i < 0$ because $y_i^{(k+1)} = -1$, then $\Delta E < 0$
    - $y_i^{(k)} = -1, y_i^{(k+1)} = +1$ case
      $\rightarrow \Delta y_i = +2$ and $net_i > 0$ because $y_i^{(k+1)} = +1$, then $\Delta E < 0$
    - $y_i^{(k)} = y_i^{(k+1)}$ case
      $\rightarrow \Delta y_i = 0$, then $\Delta E = 0$
    - thus, $\Delta E \leq 0$

Single-layer feedback networks and associative memories (계속)

- Lyapunov stability theorem
  - used to prove the stability of a dynamic system
  - Consider the autonomous (i.e., unforced) system described with a system of n first-order linear or nonlinear differential equations:

Slide 67

$$\begin{aligned} \dot{y}_1 &= f_1(y) \\ \dot{y}_2 &= f_2(y) \\ &\vdots \\ \dot{y}_n &= f_n(y) \end{aligned}$$

or in vector-matrix notation: $\dot{\mathbf{y}}(t) = \mathbf{f}(\mathbf{y})$
where $\mathbf{y}(t) = (y_1, y_2, \cdots, y_n)^T$ is the state vector and $\mathbf{f}(\mathbf{y}) = (f_1, f_2, \cdots, f_n)^T$ is a nonlinear vector function.

Single-layer feedback networks and associative memories (계속)

Slide 68

Asymptotically stable (means the state vectors converges to zero as time goes to infinity) can be accomplished if a positive-definite(energy) function $E(\mathbf{y})$ can be found such that

1. $E(\mathbf{y})$ is continuous with respect to all the components y_i for $i = 1, 2, \cdots, n$, and
2. $dE[\mathbf{y}(t)]/dt < 0$, which indicates that the energy function is decreasing in time

* the energy function $E(\mathbf{y})$ is called *Lyapunov function*
  - is not unique for a given system
  - if at least one function exists, the system is asymptotically stable

Single-layer feedback networks and associative memories (계속)

Slide 69

- associative memories
  - can store a set of patterns as memories
  - when a key pattern (test pattern) is presented, then it responds by producing most resemble or relate stored pattern
  - called
    * content-addressable memories in contrast to address-addressable memories in digital computers

Single-layer feedback networks and associative memories (계속)

Slide 70

- two types of associative memories
  - p pairs of vectors $\{(\mathbf{x}^1, \mathbf{y}^1), (\mathbf{x}^2, \mathbf{y}^2), \ldots, (\mathbf{x}^p, \mathbf{y}^p)\}$ with $\mathbf{x}^i \in R^n$ and $\mathbf{y}^i \in R^m$
  - autoassociative memory
    - $\mathbf{x}^i = \mathbf{y}^i$
    - network mapping $\Phi(\mathbf{x}^i) = \mathbf{x}^i$
    - thus, $\Phi(\mathbf{x}) = \mathbf{x}^i$ if $\mathbf{x}$ is the most closer to $\mathbf{x}^i$
  - heteroassociative memory:
    - network mapping $\Phi(\mathbf{x}^i) = \mathbf{y}^i$
    - thus, $\Phi(\mathbf{x}) = \mathbf{y}^i$ if $\mathbf{x}$ is the most closer to $\mathbf{x}^i$
- distance
  - two vectors $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$ and $\mathbf{x}^{'} = (x_1^{'}, x_2^{'}, \ldots, x_n^{'})^T$
  - Euclidean distance: $d = [(x_1 - x_1^{'})^2 + \cdots + (x_n - x_n^{'})^2]^{1/2}$
  - Hamming distance: $HD(\mathbf{x}, \mathbf{x}^{'}) = \begin{cases} \sum_{i=1}^{n} \mid x_i - x_i^{'} \mid & \text{if } x_i, x_i^{'} \in 0, 1 \\ \frac{1}{2} \sum_{i=1}^{n} \mid x_i - x_i^{'} \mid & \text{if } x_i, x_i^{'} \in -1, 1 \end{cases}$

Single-layer feedback networks and associative memories (계속)

Slide 71

- recurrent autoassociative memory—Hopfield memory
  - data retrieval rule:

$$y_i(k+1) = sgn\left(\sum_{j=1(j\neq i)}^{n} w_{ij} y_j^{(k)} + x_i - \theta_i\right), i = 1, 2, \cdots, n$$

  - finding weight matrix
    - where bipolar binary vectors $\mathbf{x}^k, k = 1, 2, \ldots, p$

$$\mathbf{W} = \sum_{k=1}^{p} \mathbf{x}^k (\mathbf{x}^k)^T - p\mathbf{I}$$

or

$$w_{ij} = \sum_{k=1}^{p} x_i^k x_j^k, i \neq j, w_{ii} = 0$$

    - where unipolar binary vectors, i.e., $x_i^k \in 0, 1$

Single-layer feedback networks and associative memories (계속)

$$w_{ij} = \sum_{k=1}^{p} (2x_i^k - 1)(2x_j^k - 1), i \neq j, w_{ii} = 0$$

- →basically Hebbian learning rule with zero initial weights
- →called Hebbian-type learning rule or outer-product learning rule

Slide 72

* ex. 11.4)

Example 11.4

[Zurada, 1992]. Consider the use of a Hopfield memory to store the two vectors $\mathbf{x}^1$ and $\mathbf{x}^2$

$$\mathbf{x}^1 = [1, -1, -1, 1]^T \quad \text{and} \quad \mathbf{x}^2 = [-1, 1, -1, 1]^T .$$

From Eq. (11.22), we obtain the weight matrix as

$$\mathbf{W} = \sum_{k=1}^{2} \mathbf{x}^k (\mathbf{x}^k)^T - 2\mathbf{I} = \begin{bmatrix} 0 & -2 & 0 & 0 \\ -2 & 0 & 0 & 0 \\ 0 & 0 & 0 & -2 \\ 0 & 0 & -2 & 0 \end{bmatrix},$$

Single-layer feedback networks and associative memories (계속)

Slide 73

and from Eq. (11.2), the energy function is

$$E(\mathbf{x}) = -\tfrac{1}{2}\mathbf{x}^T\mathbf{W}\mathbf{x} = 2(x_1x_2 + x_3x_4) .$$

The state transition diagram is shown in Fig. 11.7. There are a total of 16 states, each of which corresponds to *one vertex*. This figure shows all possible asynchronous transitions and their directions. Note that every vertex is connected only to the neighboring vertex differing by a single bit because of asynchronous transitions. In Fig. 11.7, each state is associated with its energy value. It is observed that transitions are toward lower energy values. We further find that there are two extra stable states $\bar{\mathbf{x}}^1 = [-1, 1, 1, -1]^T$ and $\bar{\mathbf{x}}^2 = [1, -1, 1, -1]^T$ in addition to the two states $\mathbf{x}^1$ and $\mathbf{x}^2$ that we want to store.

Let us consider some transition examples. Starting at the state $[1, 1, 1, 1]^T$ and with nodes updating asynchronously in ascending order, we have state transitions $[1, 1, 1, 1]^T \rightarrow [-1, 1, 1, 1]^T \rightarrow [-1, 1, 1, 1]^T \rightarrow [-1, 1, -1, 1]^T \cdots$. Hence, the state will converge at the stored pattern $\mathbf{x}^2$. However, it is possible (with a different updating order) that the state $[1, 1, 1, 1]^T$ will converge to $\mathbf{x}^1 = [1, -1, -1, 1]^T$, $\bar{\mathbf{x}}^1 = [-1, 1, 1, -1]^T$ or $\bar{\mathbf{x}}^2 = [1, -1, 1, -1]^T$. This happens because the Hamming distance between the initial state, $[1, 1, 1, 1]^T$ and any of $\mathbf{x}^1$, $\mathbf{x}^2$, $\bar{\mathbf{x}}^1$, or $\bar{\mathbf{x}}^2$ is of the same value 2.

Single-layer feedback networks and associative memories (계속)

Slide 74

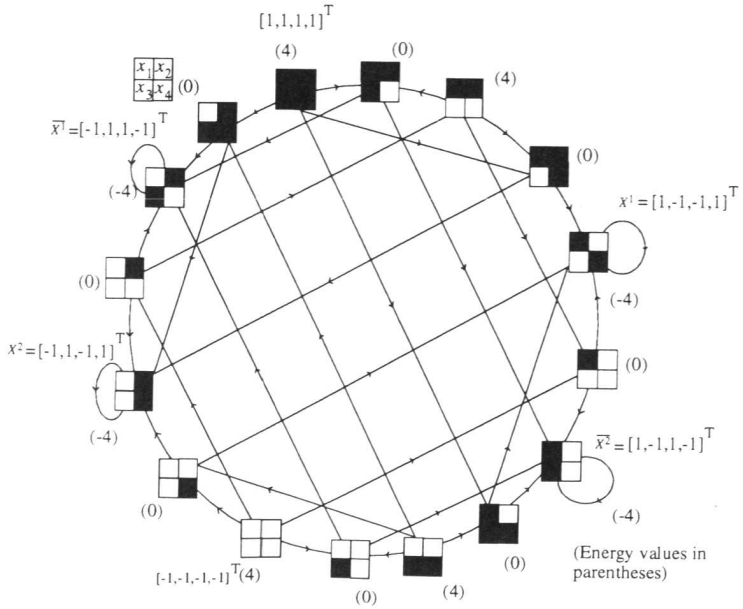
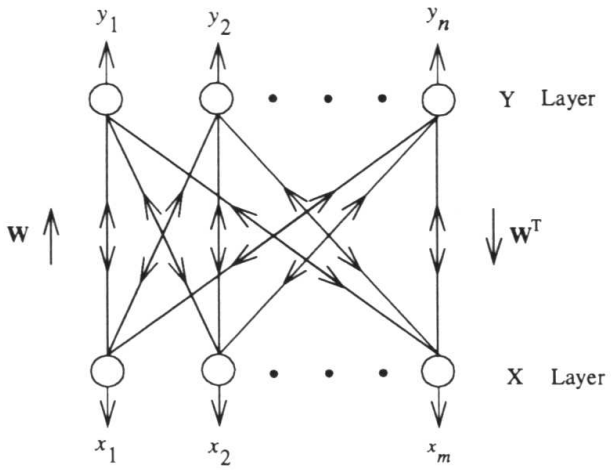


Figure 11.7 State transition diagram for Example 11.4.

Single-layer feedback networks and associative memories (계속)

Slide 75

– bidirectional associative memory (BAM)



Single-layer feedback networks and associative memories (계속)

Slide 76

* $\mathbf{y}' = a(\mathbf{W}\mathbf{x})$ or $y_i' = a\left(\sum_{j=1}^{m} w_{ij}x_j\right), i = 1, 2, \ldots, n$
* $\mathbf{x}' = a(\mathbf{W}^T\mathbf{y}')$ or $x_j' = a\left(\sum_{i=1}^{n} w_{ji}y_i'\right), j = 1, 2, \ldots, m$

$$
\begin{aligned}
\mathbf{y}^{(1)} &= a(\mathbf{W}\mathbf{x}^{(0)}) && \text{(first forward pass)}\\
\mathbf{x}^{(2)} &= a(\mathbf{W}^T\mathbf{y}^{(1)}) && \text{(first backward pass)}\\
\mathbf{y}^{(3)} &= a(\mathbf{W}\mathbf{x}^{(2)}) && \text{(second forward pass)}\\
\mathbf{x}^{(4)} &= a(\mathbf{W}^T\mathbf{y}^{(3)}) && \text{(second backward pass)}\\
&\vdots\\
\mathbf{y}^{(k-1)} &= a(\mathbf{W}\mathbf{x}^{(k-2)}) && [(k/2)\text{th forward pass}]\\
\mathbf{x}^{(k)} &= a(\mathbf{W}^T\mathbf{y}^{(k-1)}) && [(k/2)\text{th backward pass}]
\end{aligned}
$$

* consider $\{(\mathbf{x}^1, \mathbf{y}^1), (\mathbf{x}^2, \mathbf{y}^2), \ldots, (\mathbf{x}^p, \mathbf{y}^p)\}$
  where $\mathbf{x}^k = (x_1^k, x_2^k, \ldots, x_m^k)^T$ and $\mathbf{y}^k = (y_1^k, y_2^k, \ldots, y_n^k)^T$
* weight learning rule for BAM

Single-layer feedback networks and associative memories (계속)

Slide 77

$$
\mathbf{W} = \begin{cases} \sum_{k=1}^{p} \mathbf{y}^k(\mathbf{x}^k)^T & \text{for bipolar vectors} \\ \sum_{k=1}^{p} (2\mathbf{y}^k - 1)(2\mathbf{x}^k - 1)^T & \text{for unipolar vectors} \end{cases}
$$

or

$$
w_{ij} = \begin{cases} \sum_{k=1}^{p} y_i^k x_j^k & \text{for bipolar vectors} \\ \sum_{k=1}^{p} (2y_i^k - 1)(2x_j^k - 1) & \text{for unipolar vectors} \end{cases}
$$

* assume one of the stored vectors $\mathbf{x}^{k'}$ is presented to the BAM

$$
\begin{aligned}
\mathbf{y} &= a\left(\sum_{k=1}^{p} (\mathbf{y}^k(\mathbf{x}^k)^T)\mathbf{x}^{k'}\right)\\
&= a\left(n\mathbf{y}^{k'} + \sum_{k=1(k\neq k')}^{p} \mathbf{y}^k(\mathbf{x}^k)^T\mathbf{x}^{k'}\right)\\
&= a(n\mathbf{y}^{k'} + \eta)
\end{aligned}
$$

Single-layer feedback networks and associative memories (계속)

* if the vectors $\mathbf{x}^k$ are orthogonal (i.e., HD$(\mathbf{x}^k, \mathbf{x}^l) = n/2$ for $k, l = 1, 2, \ldots, p, k \neq l$), then the noise term η is zero
  $\rightarrow \mathbf{y} = \mathbf{y}^{k'}$, stabilize only a single pass
* assume one of a distorted pattern $\mathbf{x}^{k'}$ is given
  $\rightarrow$the stabilization depends on factors such as HD between $\mathbf{x}^{k'}$ and stored $\mathbf{x}^k$ vectors, the orthogonality of $\mathbf{y}^k$, and the HD between $\mathbf{y}^k$

Slide 78

* stability of BAM
  - based on Lyapunov (energy) function
  - bidirectionally stable if $\mathbf{y}^{(k)} \rightarrow \mathbf{y}^{(k+1)} \rightarrow \mathbf{y}^{(k+2)}$ and $\mathbf{y}^{(k+2)} = \mathbf{y}^{(k)}$
    energy function: $E(\mathbf{x}, \mathbf{y}) \equiv -\frac{1}{2}\mathbf{x}^T\mathbf{W}^T\mathbf{y} - \frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{x} = -\mathbf{y}^T\mathbf{W}\mathbf{x}$

$$\begin{aligned} \Delta E_{y_i} &= \nabla_{\mathbf{y}} E \Delta y_i \\ &= -\mathbf{W}\mathbf{x}\Delta y_i = -\left(\sum_{j=1}^{m} w_{ij}x_j\right)\Delta y_i, i = 1, 2, \ldots, n \end{aligned}$$

$$\Delta E_{x_j} = \nabla_{\mathbf{x}} E \Delta x_j$$

Single-layer feedback networks and associative memories (계속)

$$= -\mathbf{W}^T\mathbf{y}\Delta x_j = -\left(\sum_{j=1}^{n} w_{ji}y_i\right)\Delta x_j, j = 1, 2, \ldots, m$$

Slide 79

$$\Delta y_i = \begin{cases} 2 & \text{for } \sum_{j=1}^{m} w_{ij}x_j > 0 \\ 0 & \text{for } \sum_{j=1}^{m} w_{ij}x_j = 0 \\ -2 & \text{for } \sum_{j=1}^{m} w_{ij}x_j < 0 \end{cases}$$

$$\Delta x_j = \begin{cases} 2 & \text{for } \sum_{j=1}^{n} w_{ji}y_j > 0 \\ 0 & \text{for } \sum_{j=1}^{n} w_{ji}y_j = 0 \\ -2 & \text{for } \sum_{j=1}^{n} w_{ji}y_j < 0 \end{cases}$$

$\rightarrow$finally $\Delta E \leq 0$

* ex. 11.5)

Single-layer feedback networks and associative memories (계속)

Slide 80

Example 11.5

Given the following two pairs of vectors to be stored in a BAM:

$$\mathbf{x}^1 = [1, -1, -1, 1, 1, -1, 1, -1, -1, 1]^T, \qquad \mathbf{y}^1 = [-1, -1, 1, 1, 1, 1]^T,$$
$$\mathbf{x}^2 = [1, 1, 1, -1, 1, 1, 1, -1, 1, -1]^T, \qquad \mathbf{y}^2 = [-1, -1, 1, 1, -1, -1]^T.$$

From Eq. (11.33) we have

$$\mathbf{W} = \begin{bmatrix} -2 & 0 & 0 & 0 & -2 & 0 & -2 & 2 & 0 & 0 \\ -2 & 0 & 0 & 0 & -2 & 0 & -2 & 2 & 0 & 0 \\ 2 & 0 & 0 & 0 & 2 & 0 & 2 & -2 & 0 & 0 \\ 2 & 0 & 0 & 0 & 2 & 0 & 2 & -2 & 0 & 0 \\ 0 & -2 & -2 & 2 & 0 & -2 & 0 & 0 & -2 & 2 \\ 0 & -2 & -2 & 2 & 0 & -2 & 0 & 0 & -2 & 2 \end{bmatrix}.$$

For testing, we first choose a vector $\mathbf{x}$ with a HD of 2 from $\mathbf{x}^1$: $\mathbf{x}^{(0)} = [-1, 1, -1, 1, 1, -1, 1, -1, -1, 1]^T$. From Eq. (11.31), we obtain

$$\mathbf{y}^{(1)} = a([-4, -4, 4, 4, 8, 8]^T) = [-1, -1, 1, 1, 1, 1]^T,$$
$$\mathbf{x}^{(2)} = a([8, -4, -4, 4, 8, -4, 8, -8, -4, 4]^T) = [1, -1, -1, 1, 1, -1, 1, -1, -1, 1]^T,$$
$$\mathbf{y}^{(3)} = [-1, -1, 1, 1, 1, 1]^T = \mathbf{y}^{(1)}.$$

Hence it is bidirectionally stable and recalls the first-stored pair successfully. Let us perform another trial by choosing $\mathbf{x}^{(0)} = [-1, 1, 1, -1, 1, -1, -1, 1, 1, 1]^T$. Since $HD(\mathbf{x}^{(0)}, \mathbf{x}^1) = 7$ and $HD(\mathbf{x}^{(0)}, \mathbf{x}^2) = 5$, it is expected that the second-stored pair will be recalled. Using Eq. (11.31), we have

$$\mathbf{y}^{(1)} = a([4, 4, -4, -4, -4, -4]^T) = [1, 1, -1, -1, -1, -1]^T,$$
$$\mathbf{x}^{(2)} = a([-8, 4, 4, -4, -8, 4, -8, 8, 4, -4]^T) = [-1, 1, 1, -1, -1, 1, -1, 1, 1, -1]^T,$$
$$\mathbf{y}^{(3)} = [1, 1, -1, -1, -1, -1]^T = \mathbf{y}^{(1)}.$$

Further propagation does not change the states. The achieved stable state does not match any of the stored pairs. However, it is actually the complement of the first-stored pair: $(\mathbf{x}^{(2)}, \mathbf{y}^{(3)}) = (\bar{\mathbf{x}}^1, \bar{\mathbf{y}}^1)$. This illustrates the fact that if a pair $(\mathbf{x}, \mathbf{y})$ is stored in a BAM, then its complement $(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ is also stored.

Single-layer feedback networks and associative memories (계속)

Slide 81

- ☛ capacity : $p = \sqrt{min(m, n)}$
- ☛ can be extended multidirectional associative memory

– Boltzmann machines

* a discrete Hopfield network in which each node performs a simulated annealing process
* a state changes from $\mathbf{s}^{old}$ to $\mathbf{s}^{new}$ with probability p

$$p = \frac{1}{1 + exp(-\Delta E/T)}$$

where $\Delta E = E^{old} - E^{new}$

* stochastic update rule

$$y_i^{new} = \begin{cases} 1 & \text{if } z \le p_i = \frac{1}{1+exp(-\Delta E_i/T)} \\ y_i^{old} & \text{otherwise} \end{cases}$$

Unsupervised learning networks

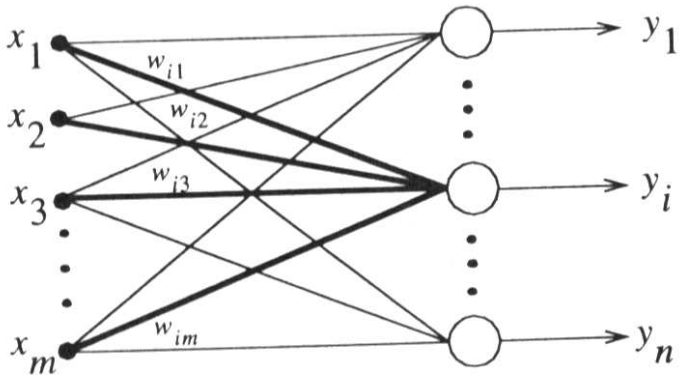
Slide 82

- unsupervised learning
  - no feedback from the environment what the desired outputs of a network
  - discover for itself any relationships such as patterns, features, correlations, or categories
  - how similar a new input pattern is to typical patterns seen in the past
    →self-organizing networks
  - useful in determining the fuzzy logic rules and fuzzy partitioning
- unsupervised learning rules in a single-layer network
  - input $\mathbf{x} = (x_1, x_2, \ldots, x_m)^T$ and output $\mathbf{y} = (y_1, y_2, \ldots, y_n)^T$
  - weight from j to node i is denoted by w_{ij}
  - $s(\cdot)$ is a monotonic nondecreasing signal function
  - Hebbian learning rule
    * $\dot{w}_{ij} = -w_{ij} + s_i(y_i)s_j(x_j)$
  - Sanger's learning rule
    * $\dot{w}_{ij} = \eta y_i \left(x_j - \sum_{k=1}^{i} y_k w_{kj}\right)$

Unsupervised learning networks (계속)

Slide 83

- Grossberg's learning rule
  * $\dot{w}_i = \eta[yx_i - w_i]u(x_i)$
- competitive learning rule



  * $\dot{w}_{ij} = s_i(y_i)[s_j(x_j) - w_{ij}]$
    where competitive signal $s_i(y_i) = \frac{1}{1+e^{-cy_i}}, c > 0$

Unsupervised learning networks (계속)

Slide 84

* with large c it acts as a binary win-lose indicator
  →"wins" if $s_i(y_i(t)) = 1$, "loses" if $s_i(y_i(t)) = 0$
  learn if win
* if we use $s_j(x_j) = x_j$, then becomes linear competitive learning rule
  $\dot{w}_{ij} = s_i(y_i)[x_j - w_{ij}]$
* competitive learning systems adaptively quantize the pattern space
* →called *adaptive vector quantization (AVQ)*
* simple but important competitive learning rules
  · called Kohonen learning rule or winner-take-all learning rule
  · the number of classes n is known a priori
  · outputs: $y_i = a(\mathbf{w}_i^T \mathbf{x})$ where $a(\cdot)$ is a continuous activation function
  · weights: $\mathbf{w}_i = (w_{i1}, w_{i2}, \ldots, w_{im})^T$
  · inputs: $\{\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^p\}$ which represents n clusters
  · learning rule
    Similarity matching : $|| \mathbf{x} - \hat{\mathbf{w}}_i^{(k)} || = \min_{1 \le j \le n} \{|| \mathbf{x} - \hat{\mathbf{w}}_j^{(k)} ||\}$
    Updating:

Unsupervised learning networks (계속)

Slide 85

$$\hat{\mathbf{w}}_i^{(k+1)} = \hat{\mathbf{w}}_i^{(k)} + \alpha^{(k)}(\mathbf{x} - \hat{\mathbf{w}}_i^{(k)})$$
$$\hat{\mathbf{w}}_j^{(k+1)} = \hat{\mathbf{w}}_j^{(k)} \qquad \text{for } j = 1, 2, \ldots, n; j \neq i$$
where $\hat{\mathbf{w}}_i = \frac{\mathbf{w}_i}{||\mathbf{w}_i||}$

· find the "winner" $\hat{\mathbf{w}}_i^{(k)}$ which is the pattern closest to the current input pattern

$$\begin{aligned} \text{argmin}_j || \mathbf{x} - \hat{\mathbf{w}}_j ||^2 &= \text{argmin}_j (\mathbf{x} - \hat{\mathbf{w}}_j)^T (\mathbf{x} - \hat{\mathbf{w}}_j) \\ &= \text{argmin}_j (\mathbf{x}^T \mathbf{x} + \hat{\mathbf{w}}^T \hat{\mathbf{w}}_j - 2\mathbf{x}^T \hat{\mathbf{w}}_j) \\ &= \text{argmax}_j (\mathbf{x}^T \hat{\mathbf{w}}_j) = \text{argmax}_j (\hat{\mathbf{w}}_j^T \mathbf{x}) \end{aligned}$$

→if the input pattern $\mathbf{x}$ correlates maximally with $\hat{\mathbf{w}}$, then
$\text{net}_i = \hat{\mathbf{w}}_i^T \mathbf{x} = || \hat{\mathbf{w}}_i || \, || \mathbf{x} || \, cos(\hat{\mathbf{w}}_i, \mathbf{x})$
→ith node wins if the input pattern $\mathbf{x}$ is more parallel to its weight vector $\hat{\mathbf{w}}$ than to any other $\hat{\mathbf{w}}_j, j \neq i$

· two factors that affect learning
1. initial weights: uniformly distributed random on unity hypersphere

Unsupervised learning networks (계속)

2. $\alpha^{(k)}$ parameters: begins with a large value and decreases gradually

☛ the sequence $\{\alpha^{(k)}; k = 0, 1, \ldots, ; 0 < \alpha^{(k)} < 1\}$ should satisfy the following conditions

Slide 86

$$\sum_{k=0}^{\infty} \alpha^{(k)} = \infty, \sum_{k=0}^{\infty} (\alpha^{(k)})^2 < \infty$$

☛ example sequences

(a) $\{\alpha^{(k)}\} = \{1, \frac{1}{2}, \frac{1}{3}, \ldots\}$

(b) $\alpha^{(k)} = \alpha_0 k^{-\beta}$(with $\beta \leq 1$)

(c) $\alpha^{(k)} = \alpha_0(1 - \beta k)$

practically $\alpha^{(k)} = 0.1(1 - k/10,000)$ for 10,000 samples $\mathbf{x}(k)$

Unsupervised learning networks (계속)

- Self-organizing feature maps
  - converts patterns of arbitrary dimensions (pattern space) into one-or two-dimensional arrays of neurons (feature space)
  - achieving dimensionality reduction
  - topology-preserving map that preserves neighborhood relations of the input pattern
  - Kohonen's self-organizing feature map

Slide 87

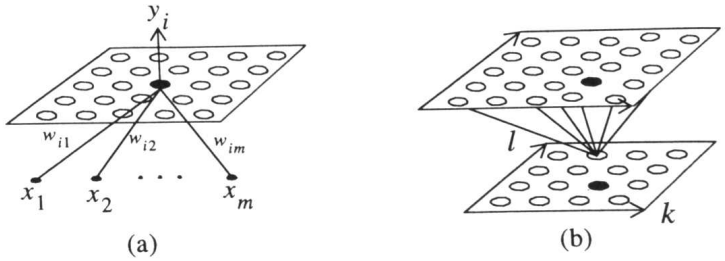


Figure 12.6 Two basic types of feature mapping networks.

Unsupervised learning networks (계속)

Slide 88

– learning rule

similarity matching:

$$|| \mathbf{x} - \hat{\mathbf{w}}_{i^*} || = \min_j \{|| \mathbf{x} - \hat{\mathbf{w}}_j ||\}$$

updating:

$$w_{ij}^{(k+1)} = \begin{cases} w_{ij}^{(k)} + \alpha^{(k)}[x_j^{(k)} - w_{ij}^{(k)}] & \text{for } i \in N_{i^*}^{(k)} \\ w_{ij}^{(k)} & \text{otherwise} \end{cases}$$

where $N_{i^*}^{(k)} = N_{i^*}(k)$ is the neighborhood set of the winner node i^* at time step k

* learning constant α and neighborhood set $N_{i^*}(k)$ is changed dynamically
* start with a wide range for $N_{i^*}(k)$ and a large α and then reduce both gradually

Unsupervised learning networks (계속)

Slide 89

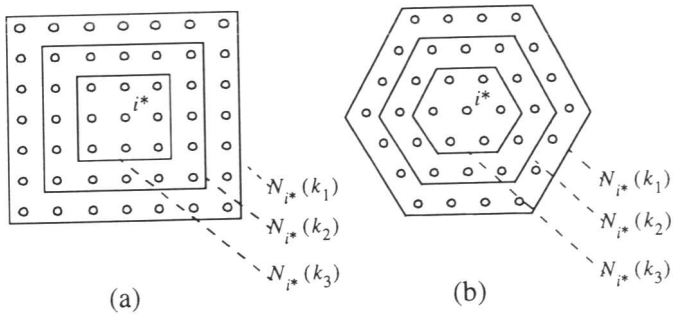


Figure 12.7 Two examples of a topological neighborhood

Adaptive Resonance Theory

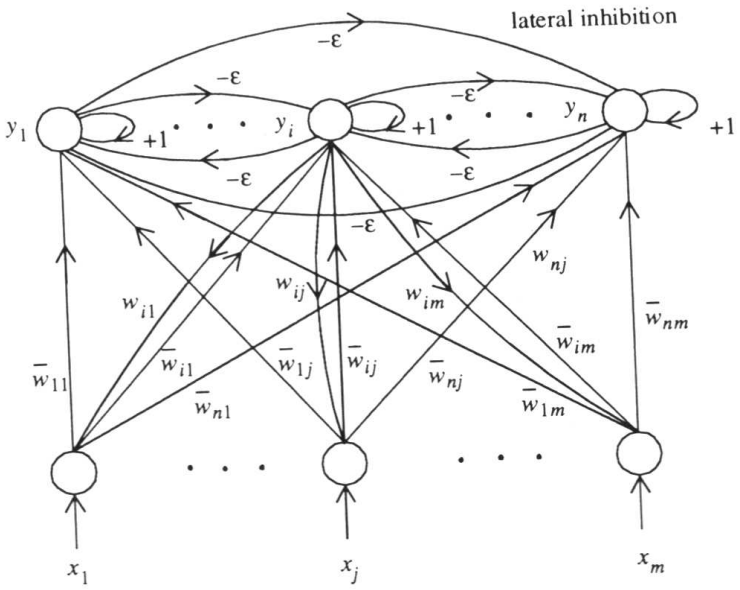
Slide 90

- ART1, ART2, and ART3 developed by Carpenter and Grossberg [1987, 1988, 1990]
- input and stored prototype are said to be resonate when they are sufficiently similar

  if not, a new node is then created to represent a new category
- sufficiently similar depends on a *vigilance* parameter ρ, with $0 < \rho \leq 1$
  - if ρ is small, resulting in a coarse categorization
  - otherwise, finely divided categories

Adaptive Resonance Theory (계속)

Slide 91



Adaptive Resonance Theory (계속)

- Algorithm ART1: Adaptive Resonance Theory for Binary Inputs

  Input: a set of pattern vector $\mathbf{x}, \mathbf{x} \in \{0,1\}^m$

  Output: A set of weight vectors $\mathbf{w}_i = (w_{i1}, w_{i2}, \ldots, w_{im})^T, i = 1, 2, \ldots, n$
  $\rightarrow n$ clusters found

step 0: initialization: set $w_{ij}(0) = 1, \bar{w}_{ij}(0) = 1/(1+m)$, for $0 < \rho \leq 1$

step 1: present a new pattern x to the input nodes

Slide 92

step 2: enable all the output nodes

step 3: use bottom-up processing to obtain a weight sum $y_i = (\bar{\mathbf{w}}_i^T)\mathbf{x} = \sum_{j=1}^m \bar{w}_{ij} x_j$
where $\bar{w}_{ij}$ is the normalization of w_{ij} given by
$\bar{w}_{ij} = \frac{w_{ij}}{\epsilon + \sum_j w_{ij}}, j = 1, 2, \ldots, m$, where (usually $\epsilon = 0.5$)

step 4: find the output node i with largest y_i value

step 5: verify that $\mathbf{x}$ belongs to the ith cluster by top-down processing
IF $r = \frac{\sum_{j=1}^m w_{ij} x_j}{||\mathbf{x}||} > \rho$, where $|| \mathbf{x} || = \sum_{j=1}^m | x_j |$
THEN $\mathbf{x}$ belongs to the ith cluster, proceed to step 6

Adaptive Resonance Theory (계속)

ELSE IF the top layer has more than a single enabled node left, then go to step 7

ELSE create a new output node i with its initial weights set as in step 0 and go to step 6

Slide 93

step 6: update the weights as follows: $w_{ij}(t+1) = w_{ij}(t) x_j, j = 1, 2, \ldots, m$ then go to step 1

step 7: the output node i is disabled by clamping y_i to 0. then go to step 3

- ex. 12-3) (see textbook)

The others

Slide 94

- the other topics
  - radial basis function networks
  - hierarchical networks–neocognitron
  - multilayer recurrent neural network
  - time-delay neural networks
  - wavelet neural networks
  - reinforcement learning algorithms