Artificial Neural Networks
Part1

Present：Shu-Wei Chang (張書維)

References:

1. Data Mining: Concepts, Models, Methods and Algorithms, Chapter 9
2. Machine learning, Chapter 4
3. MATLAB HELP
4. 類神經網路– MATLAB的應用
5. ANN class slice ,2003 Berlin Chen

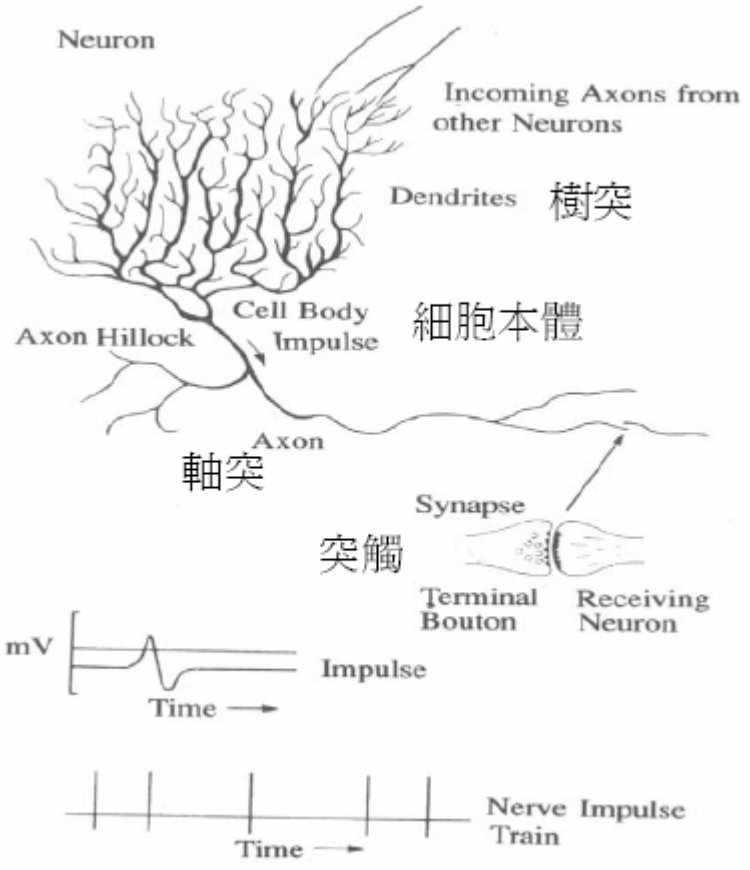
Outline

- Introduction
- Model of an artificial neuron
- Architecture of artificial neural networks
- Learning rule

Introduction

Motivation

- Human Brains
  - Neuron switching time: ~ 0.001 (10^{-3}) second
  - Number of neurons: ~10-100 billion ($10^{10} – 10^{11}$)
  - Connections per neuron: ~10-100 thousand ($10^{4} – 10^{5}$)
  - Scene recognition time: ~0.1 second
  - 100 inference steps doesn't seem sufficient! $\rightarrow$ highly parallel computation



Human Nervous System

- Ramón y Cajál (拉蒙卡哈), 1911
  - 神經解剖學者、諾貝爾獎(1906)得主
  - Introduce the idea of neurons as structural constituents of the brain
    - 推翻舊有學說
      - 腦神經系統內的細胞都是融合在一起(多核細胞體)、或細胞間細胞質流通形成神經網路
    - 提出新學說
      - 腦神經是由個別的神經元組成，在神經元內訊號以單一方向傳遞訊號(樹突→細胞本體→軸突→突觸)

神經元內/間訊號傳遞

- 樹突的主要功能就是接受其它神經元所傳遞而來的信號
- 若導致位於軸突丘的細胞膜電位超過某一特定閥值(threshold)時，則所謂的「活化電位」(action potential) 的脈衝就會被激發出來
- 藉由管狀似的軸突傳遞至其它相連接的神經元。
- 軸突的終點處是「突觸」，這種細胞間的信號傳遞以化學性的方式居多

- Def:
  - “… a system composed of many simple processing elements operating in parallel whose function is determined by network structure, connection strengths, and the processing performed at computing elements or nodes.” - DARPA (1988)
  - an artificial network is a massive parallel distributed processor made up of simple processing units. It has the ability to learn from experiential knowledge expressed through interunit connection strengths, and can make such knowledge available for use.

- Properties : (machine learning)
  - Many neuron-like threshold switching units
  - Many weighted interconnections among units
  - Highly parallel, distributed process
  - Emphasis on tuning weights automatically

- Properties and capabilities: (data mining)
  - Nonlinearity
  - Learning from examples
  - Adaptivity
  - Evidential response
  - Fault tolerance
  - Uniformity of analysis and design

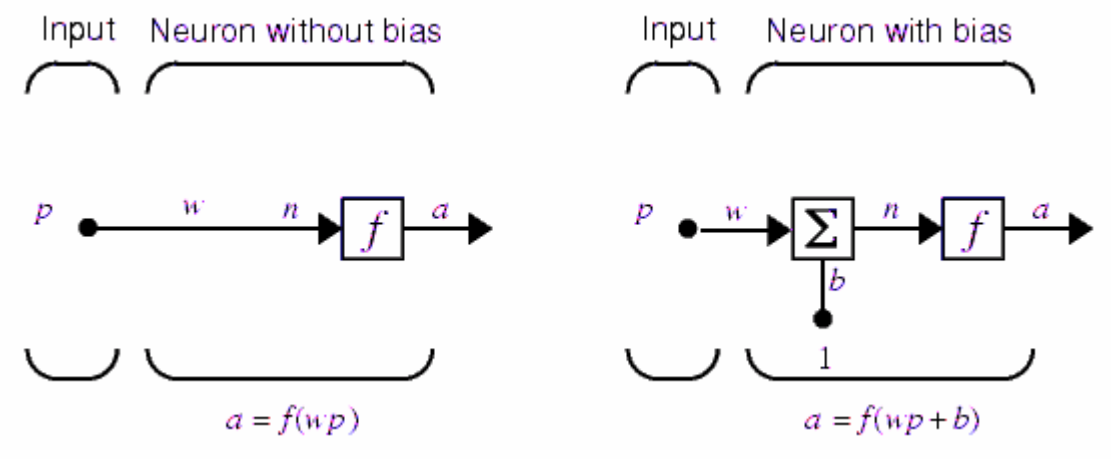
When to Consider Neural Networks

- Input: High-Dimensional and Discrete or Real-Valued
  - e.g., raw sensor input
  - Conversion of symbolic data to quantitative (numerical) representations possible
- Output: Discrete or Real Vector-Valued
  - e.g., low-level control policy for a robot actuator
  - Similar qualitative/quantitative (symbolic/numerical) conversions may apply
- Data: Possibly Noisy
- Target Function: Unknown Form
- Result: Human Readability Less Important Than Performance
  - Performance measured purely in terms of accuracy and efficiency
  - Readability: ability to explain inferences made using model; similar criteria
- Examples
  - Speech phoneme recognition [Waibel, Lee]
  - Image classification [Kanade, Baluja, Rowley, Frey]
  - Financial prediction

Model of an artificial neuron

Model of an artificial neuron

- Basic element
  - A set of connecting links
  - An adder
  - An activation function



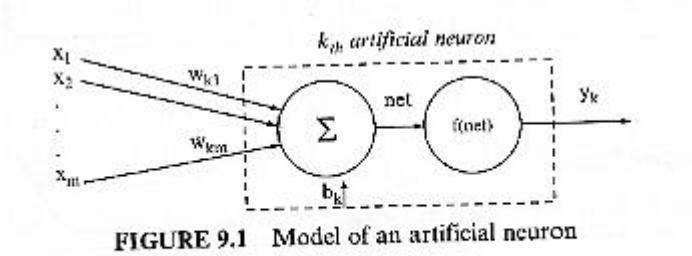
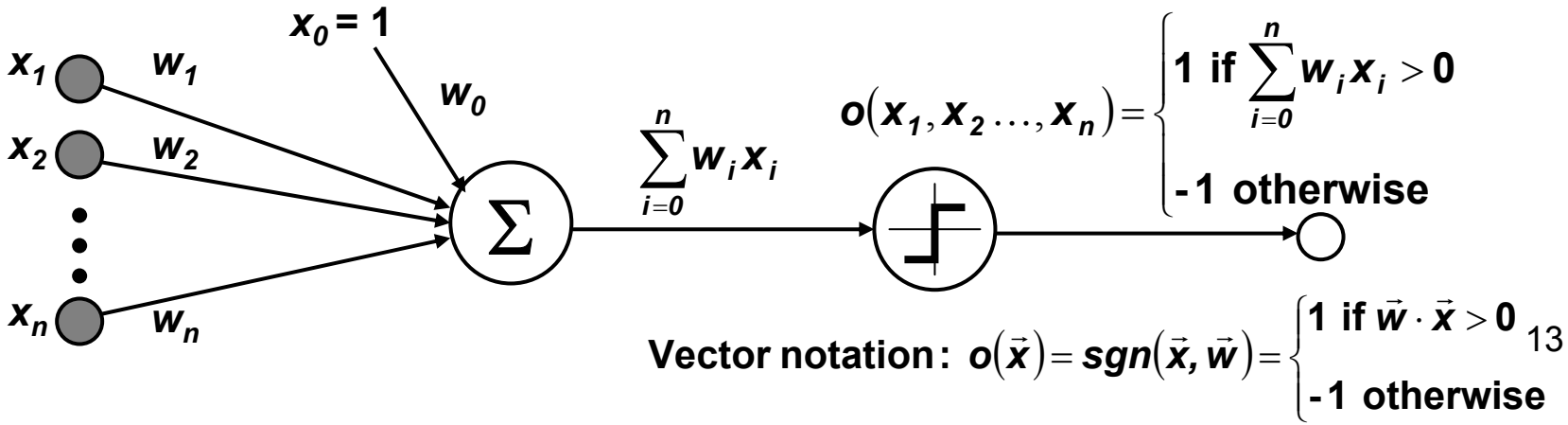


FIGURE 9.1 Model of an artificial neuron

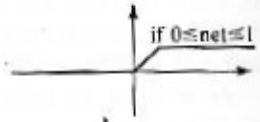
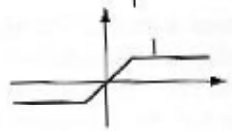
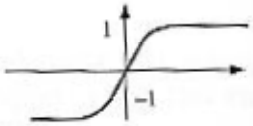


- Perceptron: Single Neuron Model
  - *aka* Linear Threshold Unit (LTU) or Linear Threshold Gate (LTG)
  - Net input to unit: defined as linear combination
  - Output of unit: threshold (activation) function on net input (threshold $\theta = w_0$)
- Perceptron Networks
  - Neuron is modeled using a unit connected by weighted links w_i to other units
  - Multi-Layer Perceptron (MLP)

A neuron's common activation functions

- Bipolar
- Unipolar

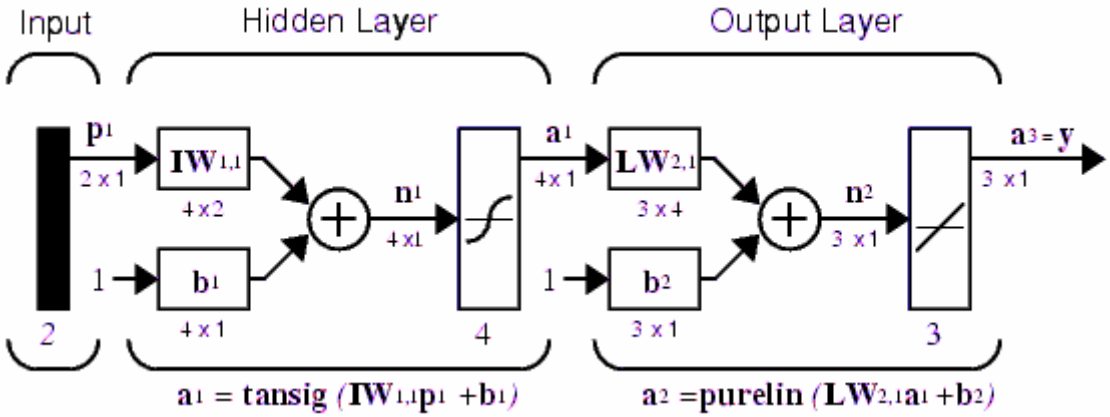
TABLE 9.1 A neuron's common activation functions

Activation Function	Input / Output Relation	Graph
Hard Limit	$y = \begin{cases} 1 & \text{if } net \geq 0 \\ 0 & \text{if } net < 0 \end{cases}$	
Symmetrical Hard Limit	$y = \begin{cases} 1 & \text{if } net \geq 0 \\ -1 & \text{if } net < 0 \end{cases}$	
Linear	$y = net$	
Saturating Linear	$y = \begin{cases} 1 & \text{if } net > 1 \\ net & \\ 0 & \text{if } net < 0 \end{cases}$	
Symmetric Saturating Linear	$y = \begin{cases} 1 & \text{if } net > 1 \\ net & \text{if } -1 \leq net \leq 1 \\ -1 & \text{if } net < -1 \end{cases}$	
Log-Sigmoid	$y = 1/(1 + e^{-net})$	
Hyperbolic Tangent Sigmoid	$y = (e^{net} - e^{-net})/(e^{net} + e^{-net})$	

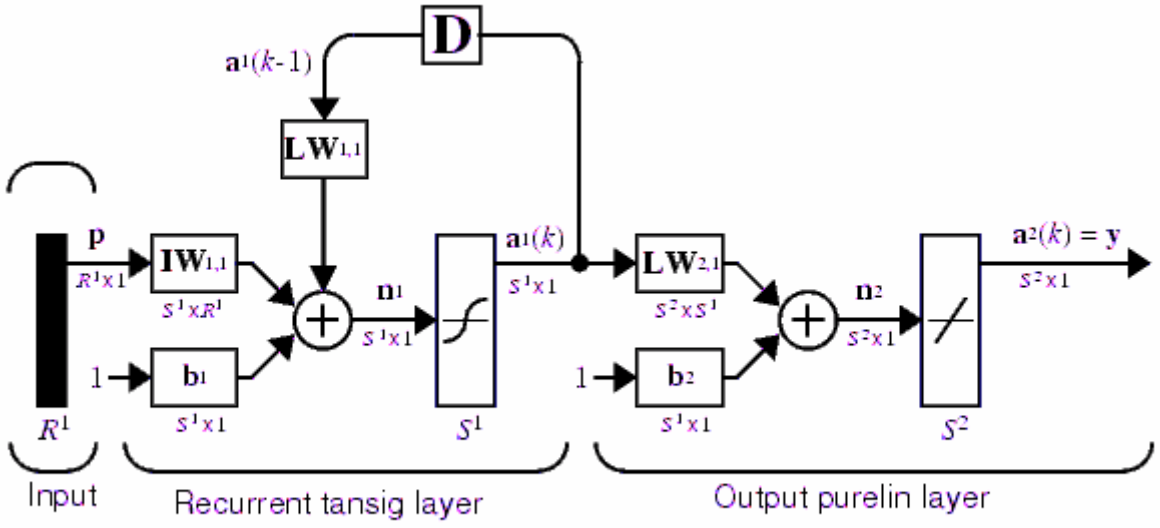
Architecture of artificial neural network

Architecture of artificial neural network

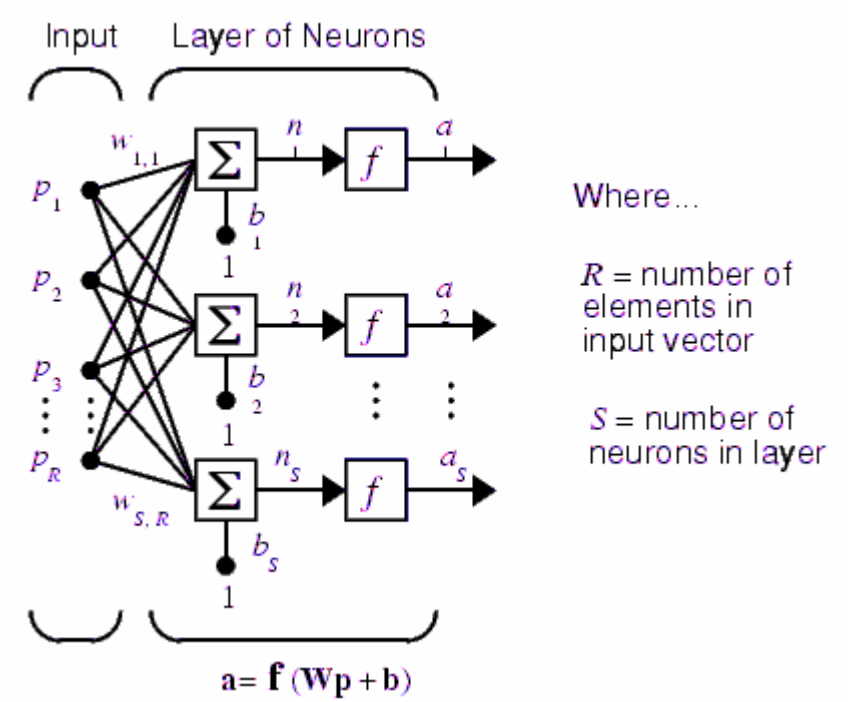
- Feedforward network
  - The process propagates from the input side to the output side unanimously, without any loops or feedback.



- Recurrent network
  - Elman Networks: This recurrent connection allows the Elman network to both detect and generate time-varying patterns.



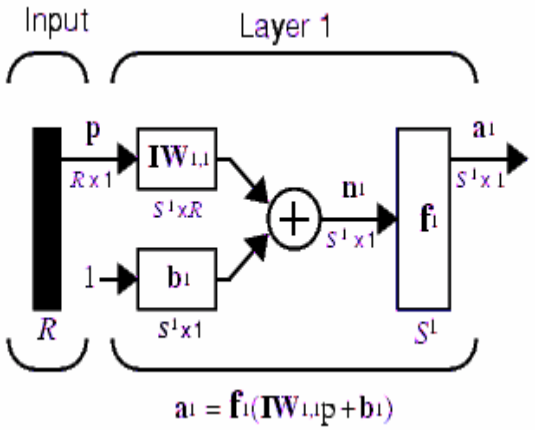
- Single layer



Where…

R = number of elements in input vector

S = number of neurons in layer

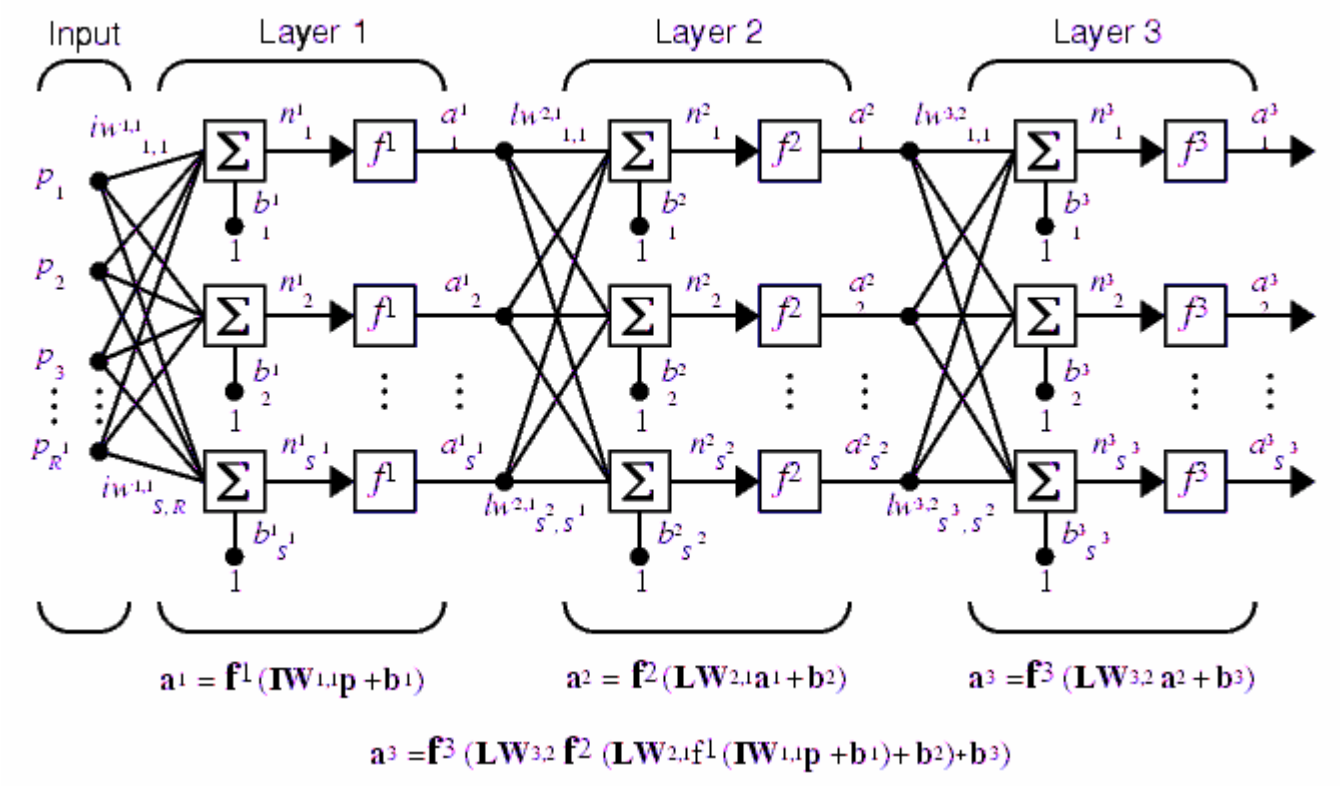


Where…

R = number of elements in input vector

S = number of neurons in Layer 1

- Multiple Layer



- Exclusive-OR(XOR) problem: Not linearly separable (cannot use a single-layer network to construct a straight line to partition the two-dimensional input space into two regions)➔ solve with a two-layer network!

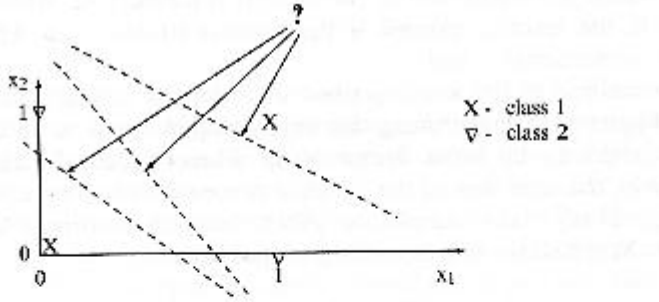


FIGURE 9.4 XOR problem

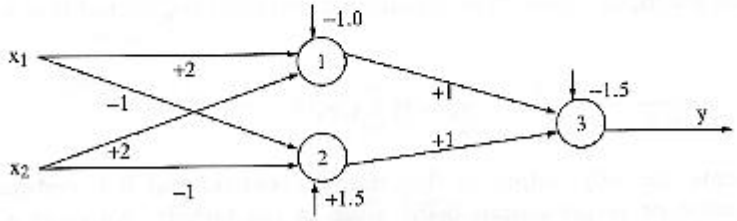


FIGURE 9.5 XOR problem solution: The two-layer ANN with the hardlimit-activation function

Learning rule

Learning rule

- Learning: a process by which the free parameters of a neural network are adapted through a process of stimulation by the environment in which the network is embedded. The type of learning is determined by the manner in which the parameters change.

- 訓練方式：
  - 逐次(Incremental)
    - 每提供一個輸入給網路就更新網路的權重值和偏權值。
  - 批次(Batch)
    - 只有在所有的輸入都提供給網路之後，才更新權重值和偏權值。
- 學習規則：
  - 監督式學習 (Supervised)
    - 提供相對應正確的輸出目標。
  - 非監督式學習 (Unsupervised)
    - 無相對應正確的輸出目標，此類演算法大部分會執行cluster運作，常在VQ中使用。

- Learning rules
  - Hebbian: strengthening connection weights when both endpoints activated
  - Perceptron: minimizing total weight contributing to errors
  - Delta Rule (LMS Rule, Widrow-Hoff): minimizing sum squared error
  - Winnow: minimizing classification mistakes on LTU with multiplicative rule
  - Correlation
  - Winner-Take-All

TABLE 2.1
Summary of learning rules and their properties.

Learning rule	Single weight adjustment Δw_{ij}	Initial weights	Learning	Neuron characteristics	Neuron / Layer
Hebbian	$co_i x_j$ $j = 1, 2, \ldots, n$	0	U	Any	Neuron
Perceptron	$c\,[d_i - \text{sgn}\,(\mathbf{w}_i^t \mathbf{x})]\,x_j$ $j = 1, 2, \ldots, n$	Any	S	Binary bipolar, or Binary unipolar*	Neuron
Delta	$c(d_i - o_i) f'(net_i) x_j$ $j = 1, 2, \ldots, n$	Any	S	Continuous	Neuron
Widrow-Hoff	$c(d_i - \mathbf{w}_i^t \mathbf{x}) x_j$ $j = 1, 2, \ldots, n$	Any	S	Any	Neuron
Correlation	$cd_i x_j$ $j = 1, 2, \ldots, n$	0	S	Any	Neuron
Winner-take-all	$\Delta w_{mj} = \alpha(x_j - w_{mj})$ m-winning neuron number $j = 1, 2, \ldots, n$	Random Normalized	U	Continuous	Layer of p neurons

c, α, β are positive learning constants
S — supervised learning, U — unsupervised learning
* — Δw_{ij} not shown

Perceptron training rule

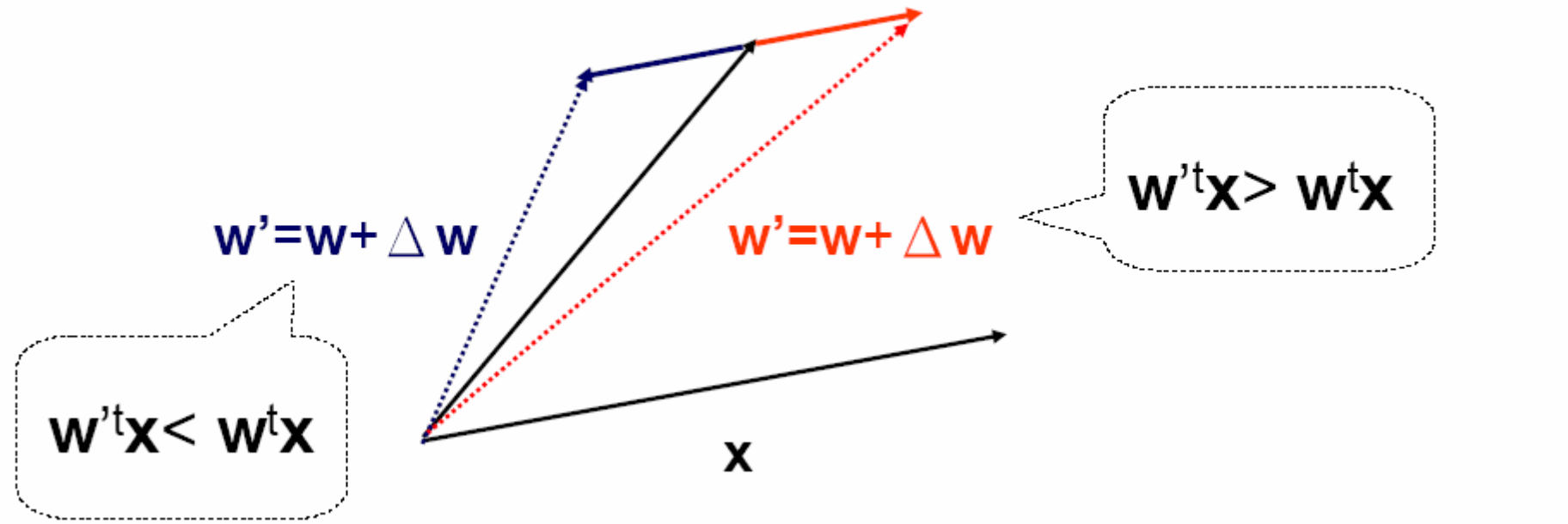
t is the target output
o is the output generated by the perceptron
η is a positive constant called the learning rate

$$w_i = w_i + \Delta w_i \qquad \Delta w_i = \eta(t-o)x_i$$

如果解答存在，感知器學習規則能夠在有限的疊代次數內收斂到解答！
可藉由正規化學習規則，減少outlier 輸入向量的敏感性。

$$\Delta w = (t-a)p^t = ep^t \rightarrow \Delta w = e\frac{p^t}{\|p\|}$$

$\triangle \mathbf{w}=-2\cdot c\cdot \mathbf{x}$ (if $\mathrm{sgn}(d_i)<\mathrm{sgn}(\mathbf{w}^t\mathbf{x})$) $\triangle \mathbf{w}=2\cdot c\cdot \mathbf{x}$ (if $\mathrm{sgn}(d_i)>\mathrm{sgn}(\mathbf{w}^t\mathbf{x})$)



- Every data sample for ANN training

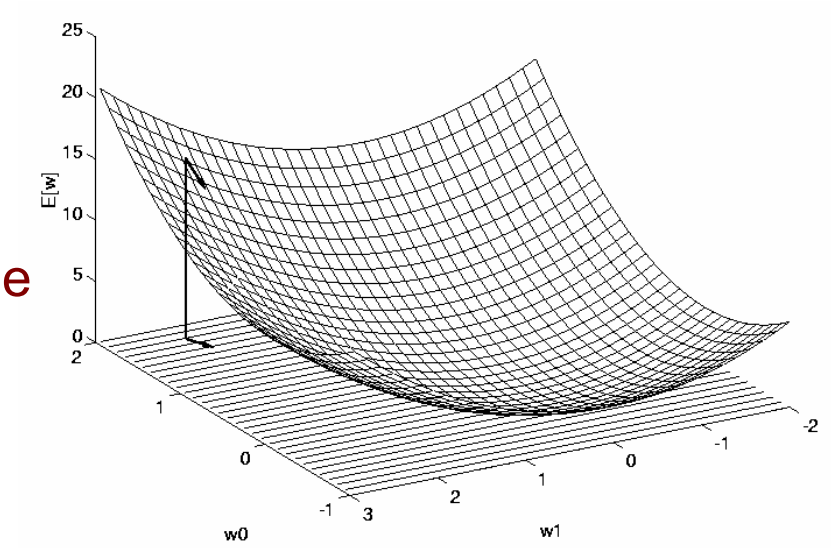
  Inputs — Output

  – Sample$_k$= x_{k1} , x_{k2},…, x_{km} d_k
  – A neuron k produces the output
    - $y_k = f(\sum_{i=1}^{m} X_i W_{ki})$
  – error $e_k(n)=d_k(n) – y_k(n)$
  – Error energy $E(n)=(1/2)e_k^2(n)$
    - How to Minimize?
      – Simple optimization
      – Move in direction of steepest gradient in weight-error space
        - Computed by finding tangent
        - i.e. partial derivatives (of *E*) with respect to weights (w_i)

Gradient Descent: Derivation of Delta/LMS (Widrow-Hoff) Rule

- Definition: Gradient

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_n} \right]$$



- Modified Gradient Descent Training Rule

$$\Delta \vec{w} = -r \nabla E[\vec{w}]$$

$$\Delta w_i = -r \frac{\partial E}{\partial w_i}$$

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \left[\frac{1}{2} \sum_{x \in D} (t(x) - o(x))^2 \right] = \frac{1}{2} \sum_{x \in D} \left[\frac{\partial}{\partial w_i} (t(x) - o(x))^2 \right]$$

$$= \frac{1}{2} \sum_{x \in D} \left[2(t(x) - o(x)) \frac{\partial}{\partial w_i} (t(x) - o(x)) \right] = \sum_{x \in D} \left[(t(x) - o(x)) \frac{\partial}{\partial w_i} (t(x) - \vec{w} \cdot \vec{x}) \right]$$

$$\frac{\partial E}{\partial w_i} = \sum_{x \in D} [(t(x) - o(x))(-x_i)]$$

Gradient Descent: Algorithm using Delta/LMS Rule

- Delta rule (LMS, Adaline, Widrow-Hoff)
  - Adjustment: $\Delta w_{kj}(n) = \eta \bullet e_k(n) \bullet x_j(n)$
  - Update value: $w_{kj}(n+1) = w_{kj}(n) + \Delta w_{kj}(n)$
  - Error-correction learning: min the cost function

- Algorithm *Gradient-Descent* (*D*, *r*)
  - Each training example is a pair of the form <*x*, *t*(*x*)>, where x is the vector of input values and *t*(*x*) is the output value. *r* is the learning rate (e.g., 0.05)
  - Initialize all weights w_i to (small) random values
  - UNTIL the termination condition is met, DO

    Initialize each Δw_i to zero

    FOR each <*x*, *t*(*x*)> in *D*, DO

    Input the instance *x* to the unit and compute the output *o*

    FOR each linear unit weight w_i, DO

    $\Delta w_i \leftarrow \Delta w_i + r(t - o)x_i$

    $w_i \leftarrow w_i + \Delta w_i$
  - RETURN final *w*

Stop criteria

- Maximum number of iterations
- Threshold level of the weight-factor may change in two consecutive iterations
- …….

Artificial Neural Networks
Part2

Present：Yao-Min Huang

Date：05/013/2004

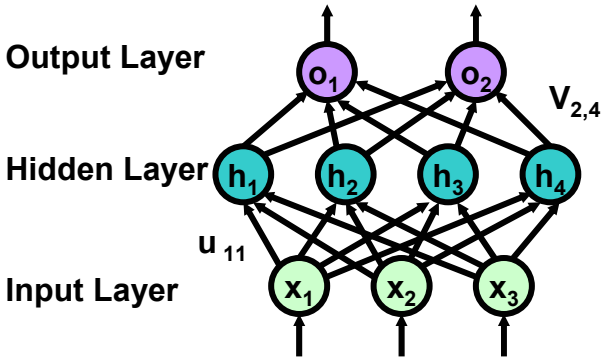
Reference：

1：Machine Learning (Tom M. Mitchell) Chapter4

2：Data Mining: Concepts, Models, Methods and Algorithms, Chapter 9

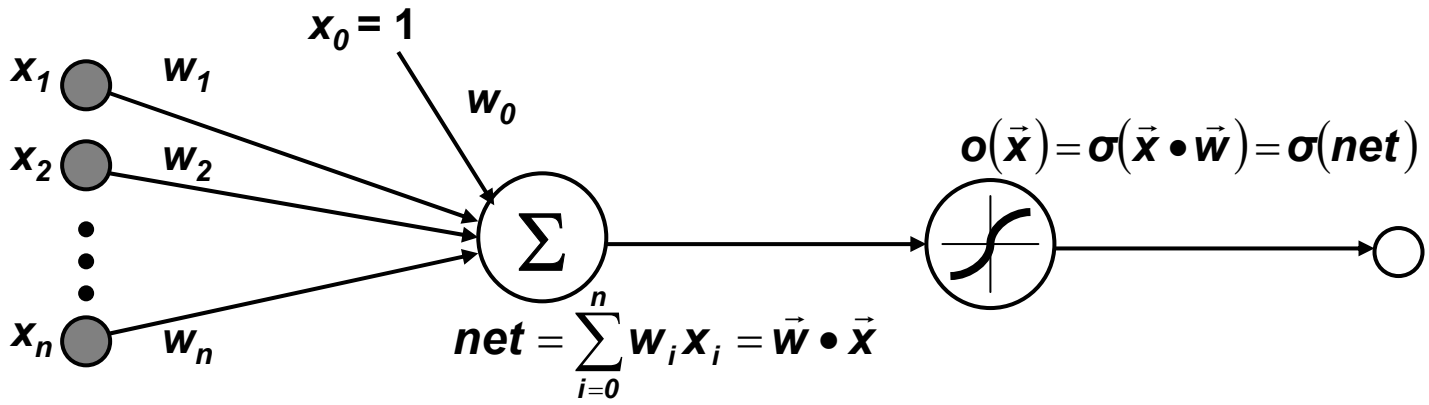
3：William H. Hsu ML & PR Lecture7

Multi-Layer Networks



- Multi-Layer Networks
  - A specific type: Multi-Layer Perceptrons (MLPs)
  - Definition: a multi-layer feedforward network is composed of an input layer, one or more hidden layers, and an output layer
  - "Layers": counted in weight layers (e.g., 1 hidden layer ≡ 2-layer network)
  - Only hidden and output layers contain perceptrons (threshold or nonlinear units)
- MLPs in Theory
  - Network (of 2 or more layers) can represent any function (arbitrarily small error)
  - Training even 3-unit multi-layer ANNs is NP-complete (Blum and Rivest, 1992)
- MLPs in Practice
  - Finding or *designing* effective networks for arbitrary functions is difficult
  - Training is very computation-intensive even when structure is "known"

Multi-Layer Networks



- **We need a unit which output is a nonlinear function of its inputs , but whose output is also a differentiable function**
  - **Sigmoid Activation Function (Squashing function)**
    - **σ is the sigmoid function** $\sigma(net) = \dfrac{1}{1+e^{-net}}$
    - **Can derive gradient rules to train**

$$\frac{d\sigma(y)}{dy} = \sigma(y) \square [1-\sigma(y)]$$

Backpropagation Rule

- Recall: Gradient of Error Function $\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots, \frac{\partial E}{\partial w_n} \right]$
- For each training example d every weight w_{ji} is update by Δw_{ji}

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

  - E_d ：error on training example d , summed over all output units

$$E_d(\vec{w}) \equiv \frac{1}{2} \sum_{k \in outputs} (t_k - o_k)^2$$

  - Weight w_{ji} can influence the rest of network only through net_j

$$\frac{\partial E_d}{\partial w_{ij}} = \frac{\partial E_d}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}} = \frac{\partial E_d}{\partial net_j} x_{ji}$$

  - $\frac{\partial E_d}{\partial net_j}$ have two cases （for Output Unit 、for Hidden Unit）

Backpropagation Rule

- For Output Unit
  - net_j can influence the rest of network by o_j

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j}\frac{\partial o_j}{\partial net_j}$$

$$= \left\{\frac{\partial}{\partial o_j}\frac{1}{2}\sum_{k\in outputs}(t_k - o_k)^2, \text{zero for all k except k=j}\right\}\{\text{derivate of the sigmoid function}\}$$

$$= \left\{\frac{\partial}{\partial o_j}\frac{1}{2}(t_j - o_j)^2\right\}\{o_j(1-o_j)\} = \left\{\frac{1}{2}2(t_j - o_j)\frac{\partial(t_j - o_j)}{\partial o_j}\right\}\{o_j(1-o_j)\}$$

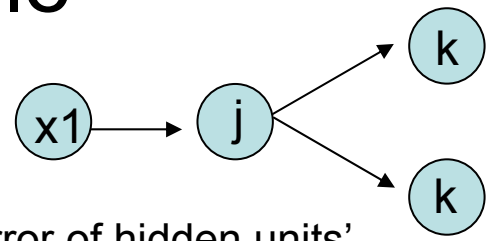
$$= \{-(t_j - o_j)\}\{o_j(1-o_j)\} = -(t_j - o_j)o_j(1-o_j)$$

  - So we have the stochastic gradient descent rule for output units

$$Define\ \delta_j = -\frac{\partial E_d}{\partial net_j} = (t_j - o_j)o_j(1-o_j)$$

$$\Delta w_{ji} = -\eta\frac{\partial E_d}{\partial w_{ji}} = \eta\delta_j x_{ji}$$

Backpropagation Rule



- For Hidden Unit
  - No target values are directly available to indicate the error of hidden units' values . Instead , the error term for hidden unit h is calculate by summing the error term for each output unit influenced by h
  - net_j can influence the rest of network by $Downstream(j)$

$$\frac{\partial E_d}{\partial net_j} = \sum_{k \in Downstream(j)} \frac{\partial E_d}{\partial net_k} \frac{\partial net_k}{\partial net_j} = \sum_{k \in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial net_j} = \sum_{k \in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial o_j} \frac{\partial o_j}{\partial net_j}$$

$$= \sum_{k \in Downstream(j)} -\delta_k w_{kj} \frac{\partial o_j}{\partial net_j} = \sum_{k \in Downstream(j)} -\delta_k w_{kj} o_j (1 - o_j)$$

$$= o_j (1 - o_j) \sum_{k \in Downstream(j)} -\delta_k w_{kj}$$

  - So we have the stochastic gradient descent rule for hidden units

$$Define\ \delta_j = -\frac{\partial E_d}{\partial net_j} = o_j (1 - o_j) \sum_{k \in Downstream(j)} \delta_k w_{kj}$$

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}} = \eta \delta_j x_{ji}$$

Backpropagation Algorithm

- Intuitive Idea: Distribute *Blame* for Error to Previous Layers
- Algorithm *Train-by-Backprop* (training samples *D*, η)
  - Each training example is a pair of the form <*x*, *t*(*x*)>, where x is the vector of input values and *t*(*x*) is the output value. η is the learning rate (e.g., 0.05)
  - Initialize all weights w_i to (small) random values
  - UNTIL the termination condition is met, DO

    FOR each <*x*, *t*(*x*)> in *D*, DO

Input the instance *x* to the unit and compute the output *o*(*x*) = σ(*net*(*x*))

FOR each output unit *k*, DO $\delta_k = o_k(x)(1-o_k(x))(t_k(x)-o_k(x))$

FOR each hidden unit h, DO $\delta_h = o_h(x)(1-o_h(x)) \sum_{k \in outputs} w_{k,h}\delta_k$

Update each $w_{j,i}$ ($w_{end\text{-}layer,start\text{-}layer}$)

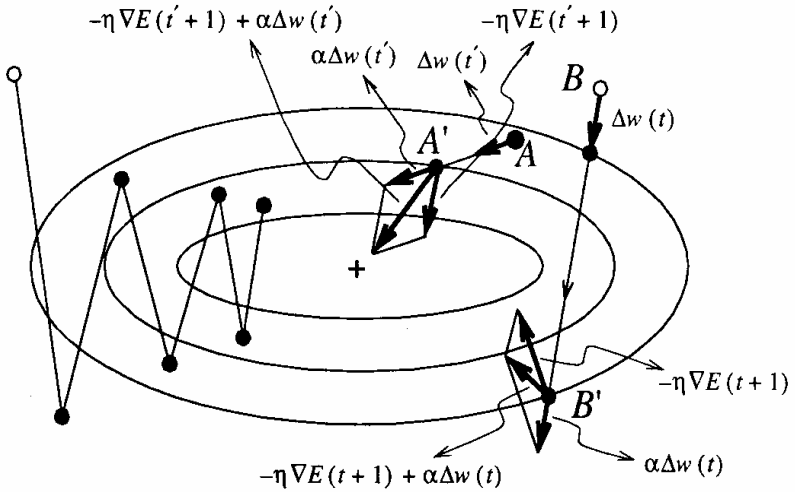
$w_{end\text{-}layer,start\text{-}layer} \leftarrow w_{end\text{-}layer,start\text{-}layer} + \Delta w_{end\text{-}layer,start\text{-}layer}$

$\Delta w_{end\text{-}layer,start\text{-}layer} \leftarrow \eta \, \delta_{end\text{-}layer} x_{end\text{-}layer,start\text{-}layer}$

Backpropagation and Local Optimal

- Gradient Descent in Backprop
  - Performed over entire *network* weight vector
  - Variety of termination condition
    - After a fixed number of iterations or the error falls below some thresold or …
  - Property: Backprop on feedforward ANNs will find a *local* (not necessarily global) error minimum
- Backprop in Practice
  - Training often *very* slow: thousands of iterations over *D*
  - Inference typically very fast
    - Classification ,Control , …
  - Alleviate the problem of local minima
    - Training multiple networks using the same data , but initializing different random weight & chose the best in validation data set
    - Adding momentum α (speeding convergence)

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

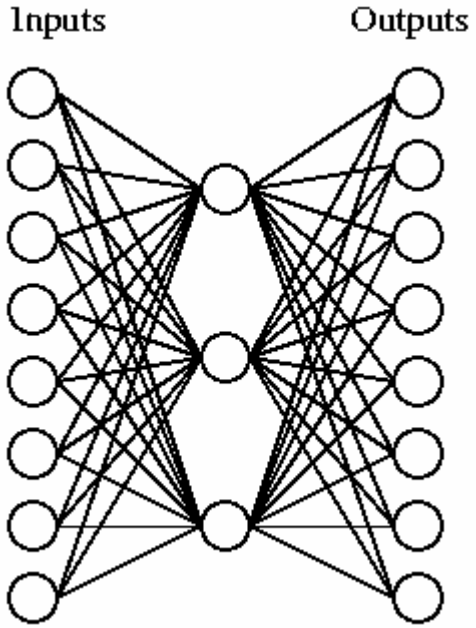


Remarks on the BackPropagation Algorithm

- Representational Power of Feedforward Network
  - 2-layer feedforward ANN
    - Any Boolean function (simulate a 2-layer AND-OR network)
    - Any bounded continuous function [Cybenko, 1989; Hornik *et al*, 1989]
  - 3-layer feedforward ANN: Arbitrary functions [Cybenko, 1988]
- Hypothesis Space Search
  - *Hypothesis space is the n-dimensional Euclidean space* (weight space)
  - Continuous, in contrast to the hypothesis spaces on decision tree learning and other methods based on discrete representation.
- Inductive Bias of ANNs
  - Preference bias: "smooth interpolation" among positive examples
  - Not well understood yet (known to be computationally hard)

Learning Hidden Layer Representations

- Hidden Units and Feature Extraction
  - Sometimes backprop will define new hidden features that are not explicit in the input representation *x*, but which capture properties of the input instances that are most relevant to learning the target function *t*(*x*)
  - Hidden units express *newly constructed features*
  - The ability of multilayer networks to automatically discover useful representations at the hidden layers Is a key feature of ANN learning.



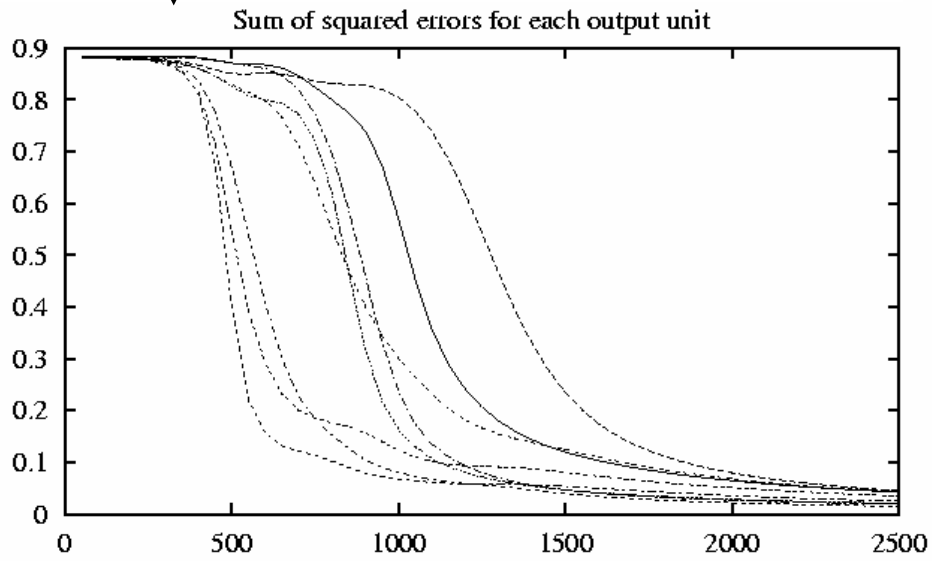
- A Target Function (Coding)

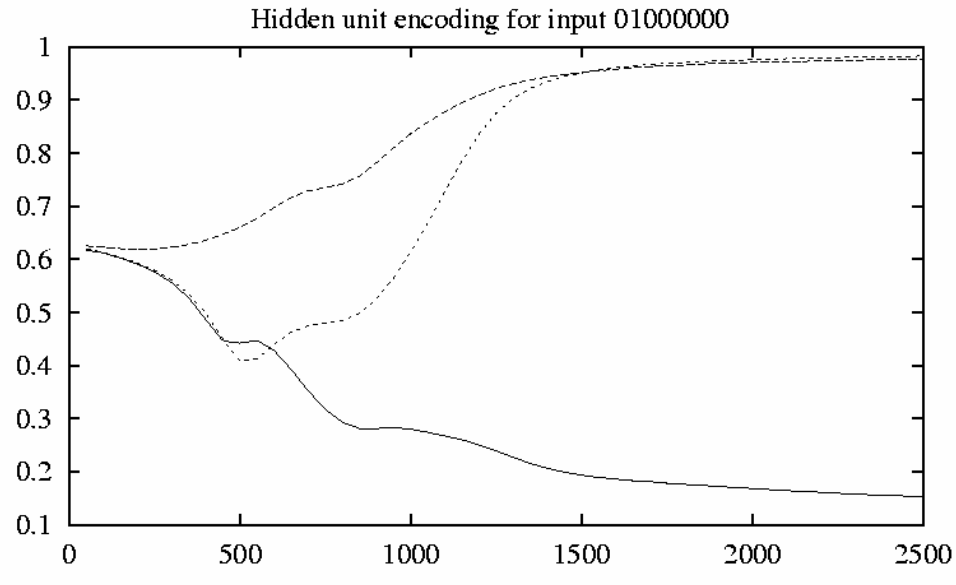
Input									Hidden Values				Output							
1	0	0	0	0	0	0	0	→	0.89	0.04	0.08	→	1	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	→	0.01	0.11	0.88	→	0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0	→	0.01	0.97	0.27	→	0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0	→	0.99	0.97	0.71	→	0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0	→	0.03	0.05	0.02	→	0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0	→	0.22	0.99	0.99	→	0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0	→	0.80	0.01	0.98	→	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1	→	0.60	0.94	0.01	→	0	0	0	0	0	0	0	1

  - Similar the standard binary encoding of eight values using three bits

Training: Evolution of Error and Hidden Unit Encoding

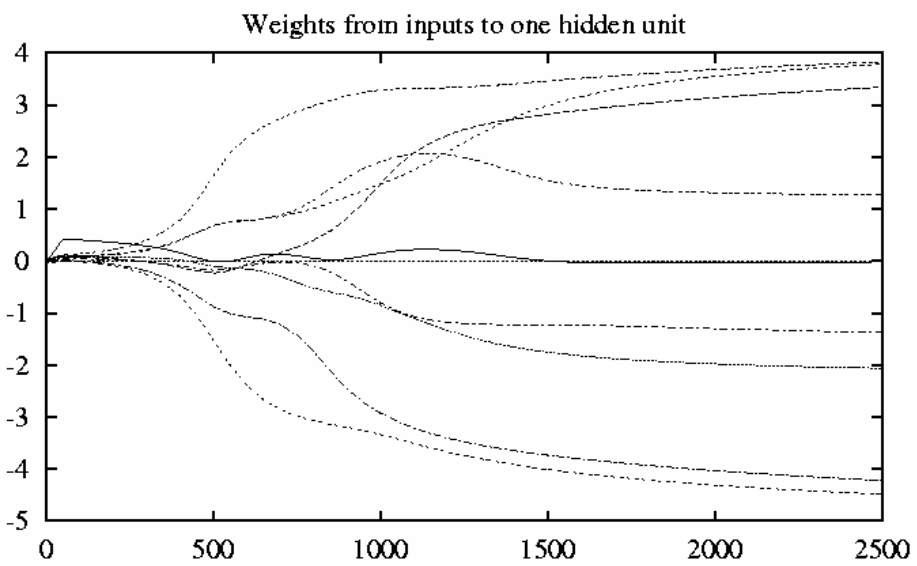
The sum of squared error for each output decreases as the gradient descent procedure proceeds





h_j(01000000), $1 \leq j \leq 3$

Training: Weight Evolution



u_{i1}, $1 \leq i \leq 8$ (which connect the eight input units)

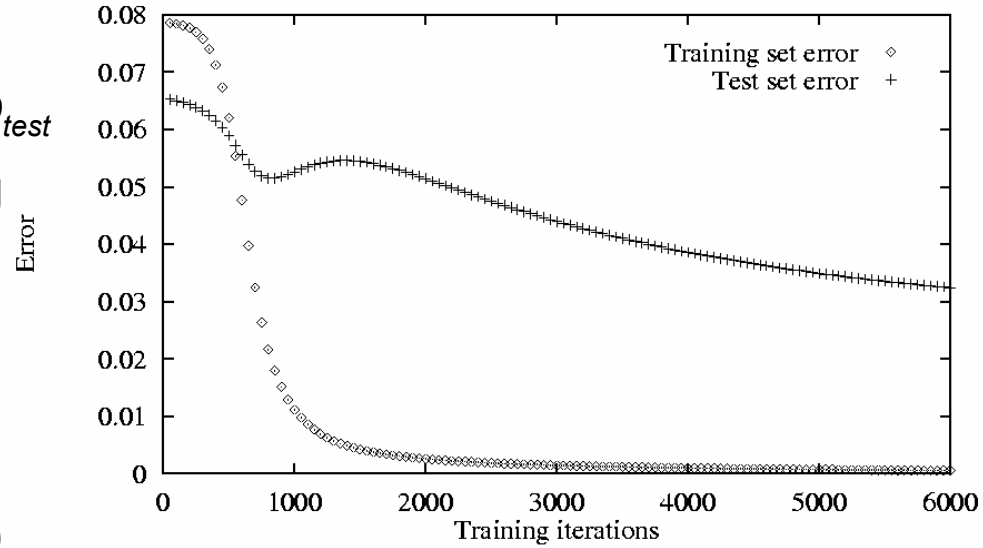
- Input-to-Hidden Unit Weights and Feature Extraction
  - Changes in first weight layer values correspond to changes in hidden layer encoding and consequent output squared errors
  - w_0 (bias weight, analogue of threshold in LTU) converges to a value near 0
  - Several changes in first 1000 epochs (different encodings)

Convergence of Backpropagation

- No Guarantee of Convergence to Global Optimum Solution
  - Gradient descent to some local error minimum (perhaps not global minimum…)
  - Possible improvements on backprop (BP)
    - Momentum term (BP variant with slightly different weight update rule)
    - Stochastic gradient descent (BP algorithm variant)
    - Train multiple nets with different initial weights; find a good mixture
  - Improvements on feedforward networks
    - Bayesian learning for ANNs (e.g: simulated annealing [Metropolis, 1953])
    - Other global optimization methods that integrate over multiple networks

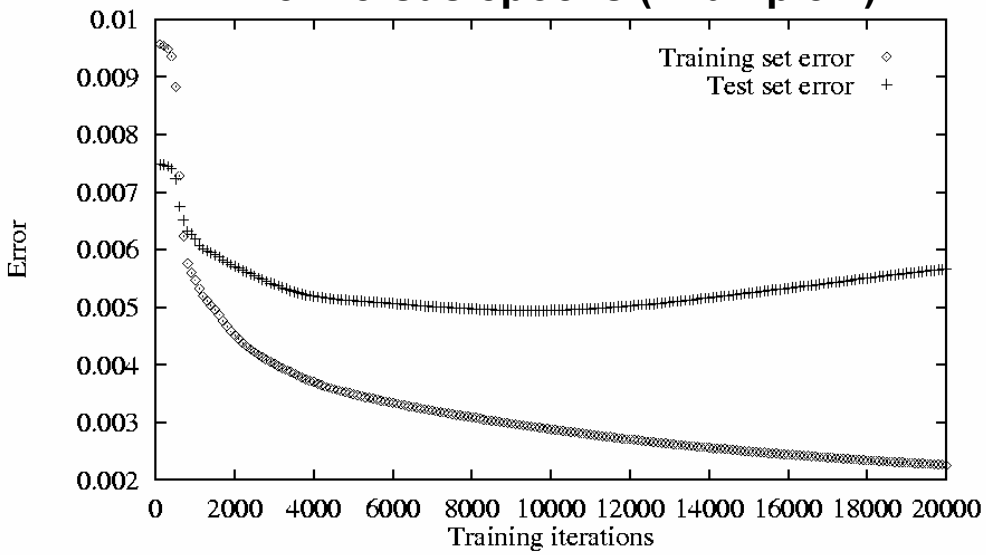
Overtraining in ANNs

- Recall: Definition of Overfitting
  - *h'* worse than *h* on D_{train}, better on D_{test}
- Overtraining: A Type of Overfitting
  - Due to excessive iterations
  - Avoidance: stopping criterion (cross-validation: holdout, *k*-fold)
  - Avoidance: weight decay



Error versus epochs (Example 2)

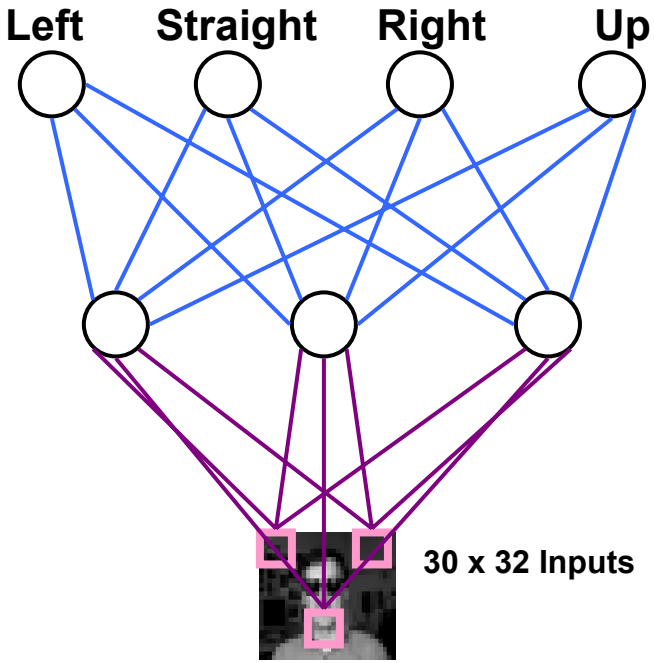
Error versus epochs (Example 1)

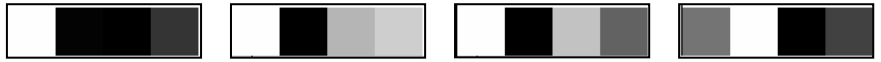


Overfitting in ANNs

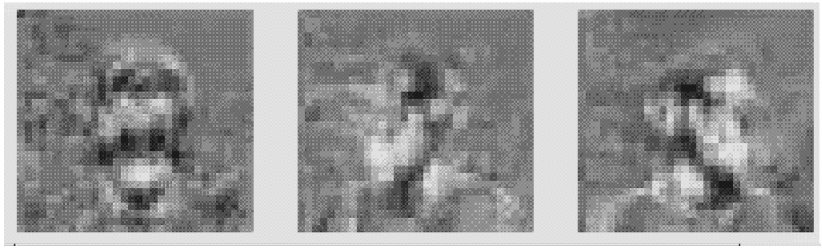
- Other Causes of Overfitting Possible
  - Number of hidden units sometimes set in advance
  - Too few hidden units ("underfitting")
    - ANNs with no growth
    - Analogy: underdetermined linear system of equations (more unknowns than equations)
  - Too many hidden units
    - ANNs with no pruning
    - Analogy: fitting a quadratic polynomial with an approximator of degree >> 2
- Solution Approaches
  - Prevention: attribute subset selection (using pre-filter or wrapper)
  - Avoidance
    - Hold out cross-validation (CV) set or split *k* ways
    - Weight decay: decrease each weight by some factor on each epoch
  - Detection/recovery: random restarts, addition and deletion of weights, units

Example: Neural Nets for Face Recognition

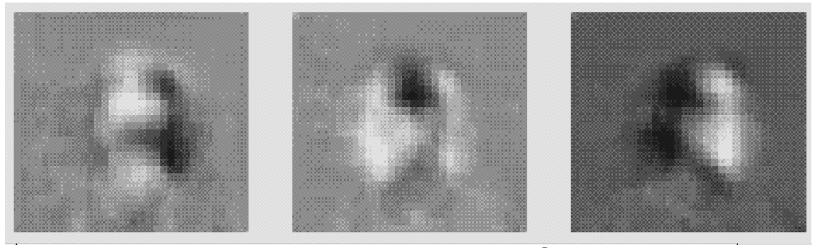




Output Layer Weights (including $w_0 = \theta$) after 1 Epoch



Hidden Layer Weights after 25 Epochs



Hidden Layer Weights after 1 Epoch

- 90% Accurate Learning Head Pose, Recognizing 1-of-20 Faces
- http://www.cs.cmu.edu/~tom/faces.html

Alternative Error Functions

- Penalize Large Weights (with Penalty Factor γ)

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{\langle \vec{x}, t(\vec{x}) \rangle \in D} \sum_{k \in outputs} \left[\left(t_k(\vec{x}) - o_k(\vec{x}) \right)^2 + \gamma \sum_{start-layer, end-layer} w^2_{end-layer, start-layer} \right]$$

  - Reduce the risk of overfitting
  - Identical to the Backprop rule , except that each weight is multiplied by the constant $(1 - 2\gamma\eta)$

- Train on Both Target Slopes and Values

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{\langle \vec{x}, t(\vec{x}) \rangle \in D} \sum_{k \in outputs} \left[\left(t_k(\vec{x}) - o_k(\vec{x}) \right)^2 + w_s \sum_{i \in inputs} \left(\frac{\partial t_k(\vec{x})}{\partial x_i} - \frac{\partial o_k(\vec{x})}{\partial x_i} \right)^2 \right]$$

- Minimizing the cross entropy

$$-\sum_{d \in D} t_d \log o_d + (1 - t_d) \log(1 - o_d)$$

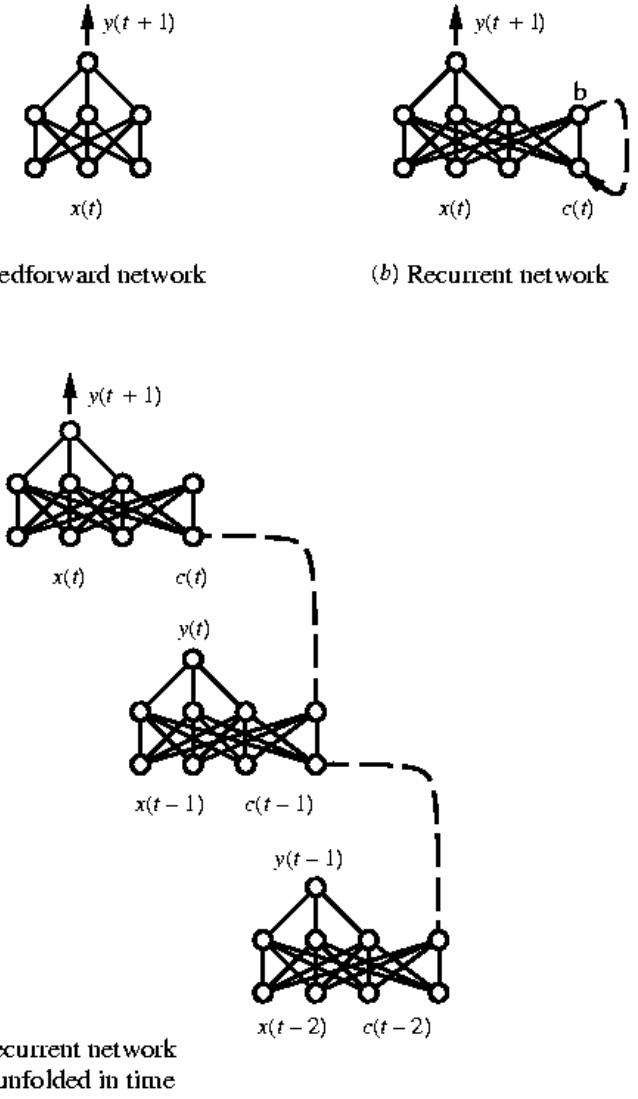
  - Wish for the network to output probability estimates

Alternative Error Minimization Procedures

- Two decision
  - Choosing a direction in which to alter the current weight vector
  - Choosing a distance to move
- Backprop
    - Direction：by taking the negative of the gradient
    - Distance：by the learning rate constant
- Alternative
    - Line Search
      - Distance：by finding the minimum of the error function along this line
    - Conjugate gradient
      - A sequence of line searches is performed

Recurrent Networks

- Representing Time Series with ANNs
  - Feedforward ANN: $y(t + 1) = net\ (x(t))$
  - Need to capture temporal relationships
- Solution Approaches
  - Directed cycles
  - Feedback
    - Output-to-input [Jordan]
    - Hidden-to-input [Elman]
    - Input-to-input
  - Captures time-lagged relationships
    - Among $x(t' \leq t)$ and $y(t + 1)$
    - Among $y(t' \leq t)$ and $y(t + 1)$
  - Learning with recurrent ANNs
    - Elman, 1990; Jordan, 1987
    - Principe and deVries, 1992
    - Mozer, 1994; Hsu and Ray, 1998



Dynamically Modifying Network Structure

- Begin with a network containing no hidden units, then grow the network as needed by adding hidden units until the training error is reduced to some acceptable level
  - Cascade-Correlation Algorithm [Fahlman an Lebiere 1990]
- Begin with a complex network and prune it as we find that certain connections are inessential.
  - [LeCun et al. 1990]

Competitive network & Competitive learning

- Intuitive Idea: Competitive Mechanisms for Unsupervised Learning
  - A class of recurrent networks
  - The output neurons compete among themselves to become active (fired)
  - Global organization from local, competitive weight update
    - Basic principle expressed by Von der Malsburg
  - Previous work: Hebb, 1949; Rosenblatt, 1959; Von der Malsburg, 1973; Fukushima, 1975; Grossberg, 1976; Kohonen, 1982
- Three basic elements for a competitive learning rule
  - Start with identical ("neural") processing units, with random initial parameters
  - Set limit on "activation strength" of each unit
  - Allow units to compete for right to respond to a set of inputs (winner-takes-all neuron)

Competitive network & Competitive learning

- **lateral inhibition**
  - Each neuron tending to inhibit the neuron to which it is laterally connected

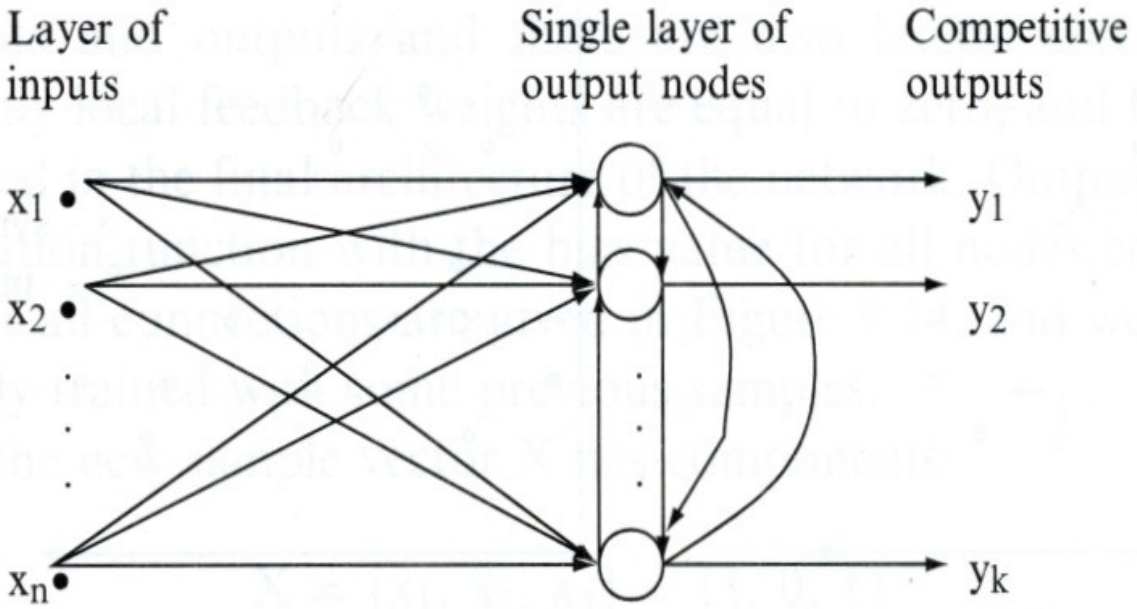


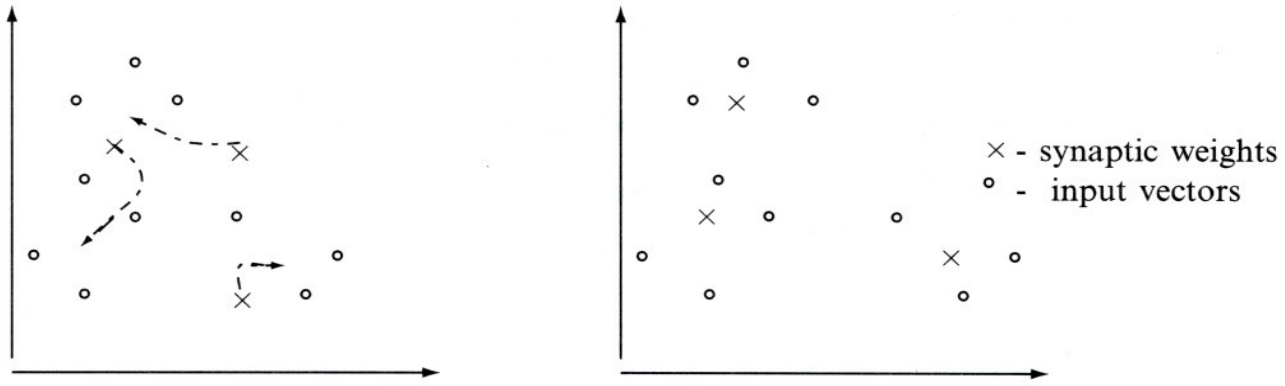
FIGURE 9.12 A graph of a simple competitive network architecture

Competitive network & Competitive learning

- The winner neuron k

$$y_k = \begin{cases} 1 & \text{if } \text{net}_k > \text{net}_j \quad \text{for all j , j} \neq \text{k} \\ 0 & \text{otherwise} \end{cases}$$

$$\Delta w_{kj} = \begin{cases} \eta(xj - w_{kj}) & \text{if neuron k wins the competition} \\ 0 & \text{if neuron k loses the competition} \end{cases}$$



a) Initial state of the network b) Final state of the network

FIGURE 9.13 Geometric interpretation of competitive learning

Competitive network & Competitive learning

- Competitive-neuron network for cluster
  - Kohonen's learning vector quantization (LVQ)
  - Self-organization Map (SOM)
  - Adaptive-resonance theory model
  - Hamming network
    - First layer：feedforward layer and it perform a correlation between the input vector & the preprocessed output vector
    - Second layer：perform a competition & the index of the second-layer neuron with a stable , positive output is the index of the prototype vector that best matches the input
- Problem
  - η Trade-off between speed of learning an the stability of the final weight factors
  - Has as many clusters as it has output neurons ➔ must know the number of clusters

Competitive network & Competitive learning

- Trace example
  - 3-dimension input , assume the network is already trained

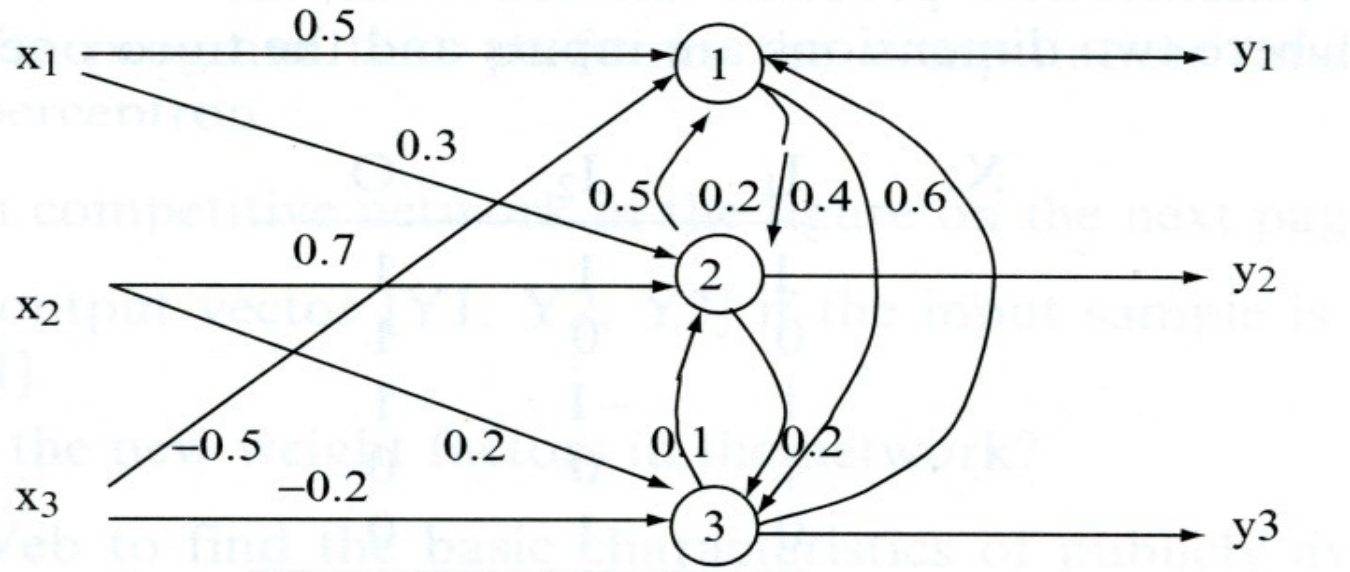


FIGURE 9.14 Example of a competitive neural network

Competitive network & Competitive learning

Given $X = \{x_1, x_2, x_3\} = \{1, 0, 1\}$ & Given the learning rate η=0.2

$$net_1^* = 0.5 \cdot x_1 + (-0.5) \cdot x_3 = 0.5 \cdot 1 - 0.5 \cdot 1 = 0$$

$$net_2^* = 0.3 \cdot x_1 + 0.7 \cdot x_2 = 0.3 \cdot 1 - 0.7 \cdot 0 = 0.3$$

$$net_3^* = 0.2 \cdot x_2 + (-0.2) \cdot x_3 = 0.2 \cdot 0 - 0.2 \cdot 1 = -0.2$$

$$net_1^* = net_1^* + 0.5 \cdot 0.3 + 0.6 \cdot (-0.2) = 0.03$$

$$net_2^* = net_2^* + 0.2 \cdot 0 + 0.1 \cdot (-0.2) = 0.28 \quad \text{(maximum !!)}$$

$$net_3^* = net_3^* + 0.4 \cdot 0 + 0.2 \cdot 0.3 = -0.14$$

So we have the output Y=$\{y_1, y_2, y_3\} = \{0, 1, 0\}$

$$\Delta w_{12} = 0.3 + 0.2(1 - 0.3) = 0.44$$

$$\Delta w_{22} = 0.7 + 0.2(0 - 0.7) = 0.56$$

$$\Delta w_{32} = 0.0 + 0.2(1 - 0.0) = 0.20$$

New Neuronal Models

- Neurons with State
  - Neuroids [Valiant, 1994]
  - Each basic unit may have a state
  - Each may use a different update rule (or compute differently based on state)
  - Adaptive model of network
    - Random graph structure
    - Basic elements receive meaning as part of learning process
- Pulse Coding
  - Spiking neurons [Maass and Schmitt, 1997]
  - Output represents more than activation level
  - Phase shift between firing sequences counts and adds expressivity
- New Update Rules
  - Non-additive update [Stein and Meredith, 1993; Seguin, 1998]
  - Spiking neuron model

Some Current Issues and Open Problems in ANN Research

- Hybrid Approaches
  - Incorporating knowledge and analytical learning into ANNs
    - Knowledge-based neural networks [Flann and Dietterich, 1989]
    - Explanation-based neural networks [Towell *et al*, 1990; Thrun, 1996]
  - Combining uncertain reasoning and ANN learning and inference
    - Probabilistic ANNs
    - Bayesian networks [Pearl, 1988; Heckerman, 1996; Hinton *et al*, 1997]
- Global Optimization with ANNs
  - Markov chain Monte Carlo (MCMC) [Neal, 1996] - e.g., simulated annealing
  - Relationship to genetic algorithms - later
- Understanding ANN Output
  - Knowledge extraction from ANNs
    - Rule extraction
    - Other decision surfaces
  - Decision support and KDD applications [Fayyad *et al*, 1996]
- Many, Many More Issues (Robust Reasoning, Representations, etc.)

Thanks for your attention!