

# **RETI NEURALI: Generalità**

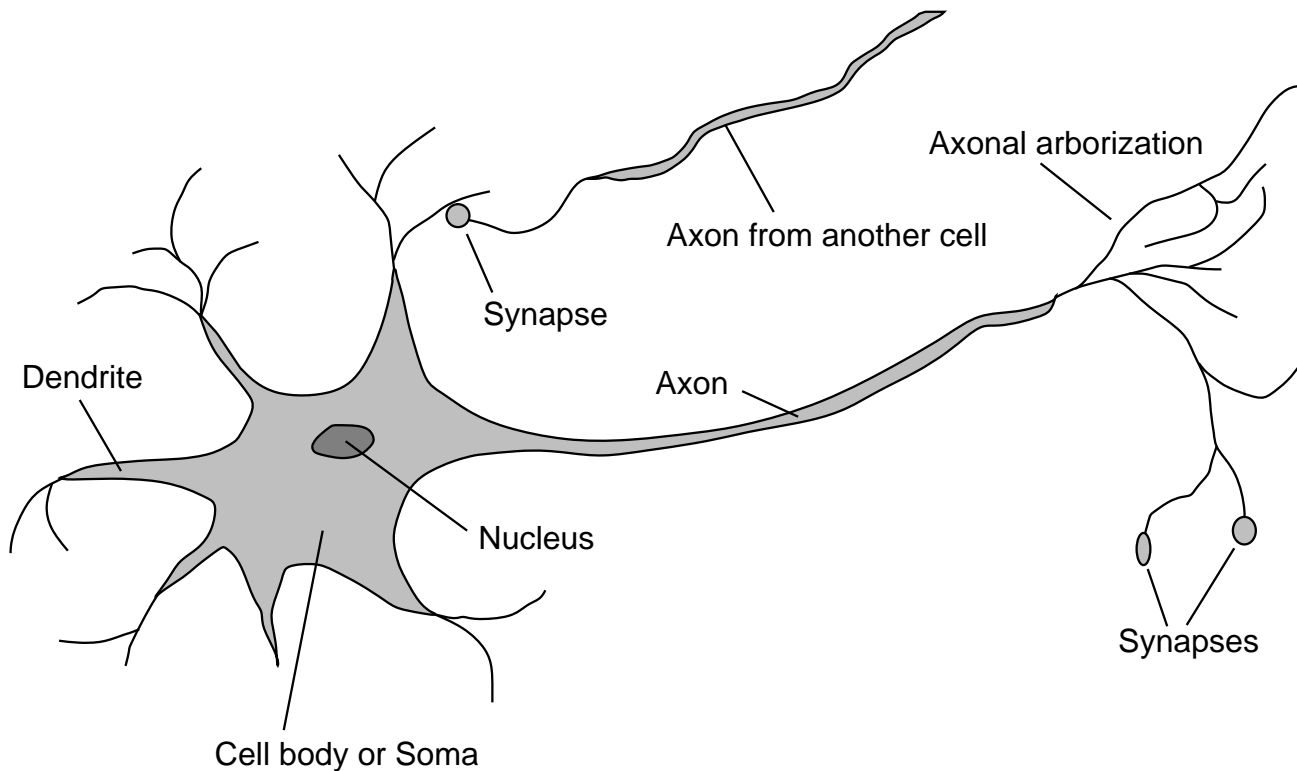
- **Tesi forte dell' AI**
  - Paradigma Connessionista
- **Modalità di funzionamento**
  - Fase di addestramento
  - Modalità operativa

## Potted history of AI

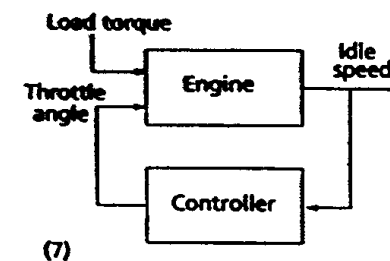
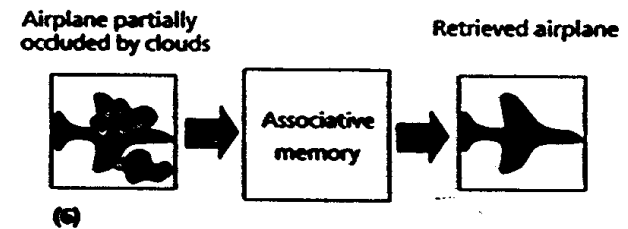
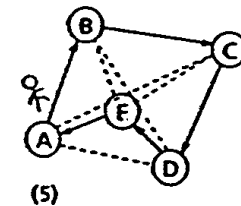
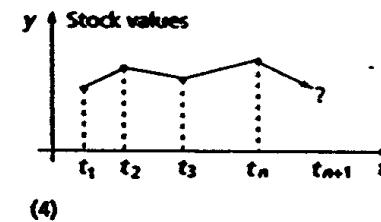
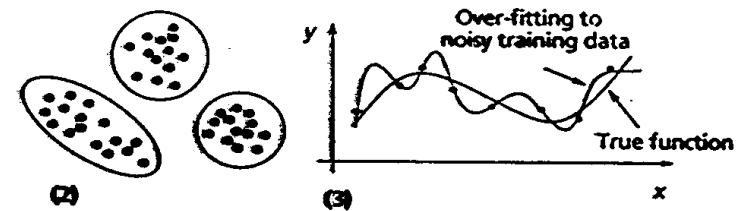
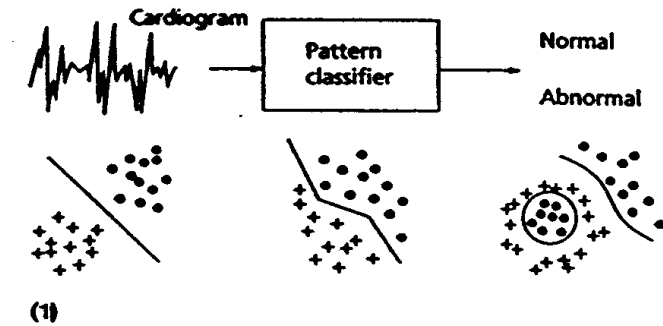
- 1943 McCulloch & Pitts: Boolean circuit model of brain
- 1950 Turing's "Computing Machinery and Intelligence"
- 1952–69 Look, Ma, no hands!
- 1950s Early AI programs, including Samuel's checkers program,  
Newell & Simon's Logic Theorist, Gelernter's Geometry Engine
- 1956 Dartmouth meeting: "Artificial Intelligence" adopted
- 1965 Robinson's complete algorithm for logical reasoning
- 1966–74 AI discovers computational complexity  
Neural network research almost disappears
- 1969–79 Early development of knowledge-based systems
- 1980–88 Expert systems industry booms
- 1988–93 Expert systems industry busts: "AI Winter"
- 1985–95 Neural networks return to popularity
- 1988– Resurgence of probability; general increase in technical depth  
"Nouvelle AI": ALife, GAs, soft computing
- 1995– Agents, agents, everywhere . . .
- 2003– Human-level AI back on the agenda

# Brains

$10^{11}$  neurons of  $> 20$  types,  $10^{14}$  synapses, 1ms–10ms cycle time  
Signals are noisy “spike trains” of electrical potential



- Pattern Classification
- Clustering/categorization
- Function approximation
- Prediction/forecasting
- Optimization
- Content-addressable memory
- Control



# TRAINING E LEARNING

- **Modalità di addestramento**

- Supervised
- Graded (o reinforcement)
- Unsupervised

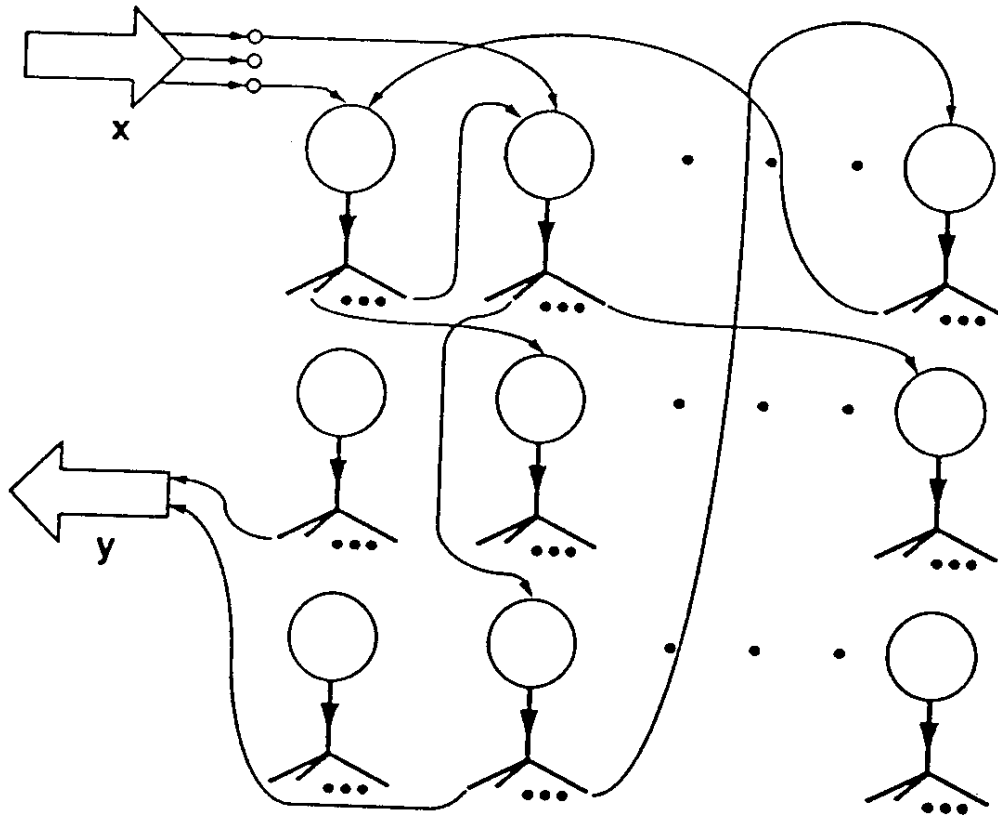
- **Leggi di apprendimento**

- Coincidence learning
- Competitive learning
- Performance learning
- Filter learning
- Spatiotemporal learning

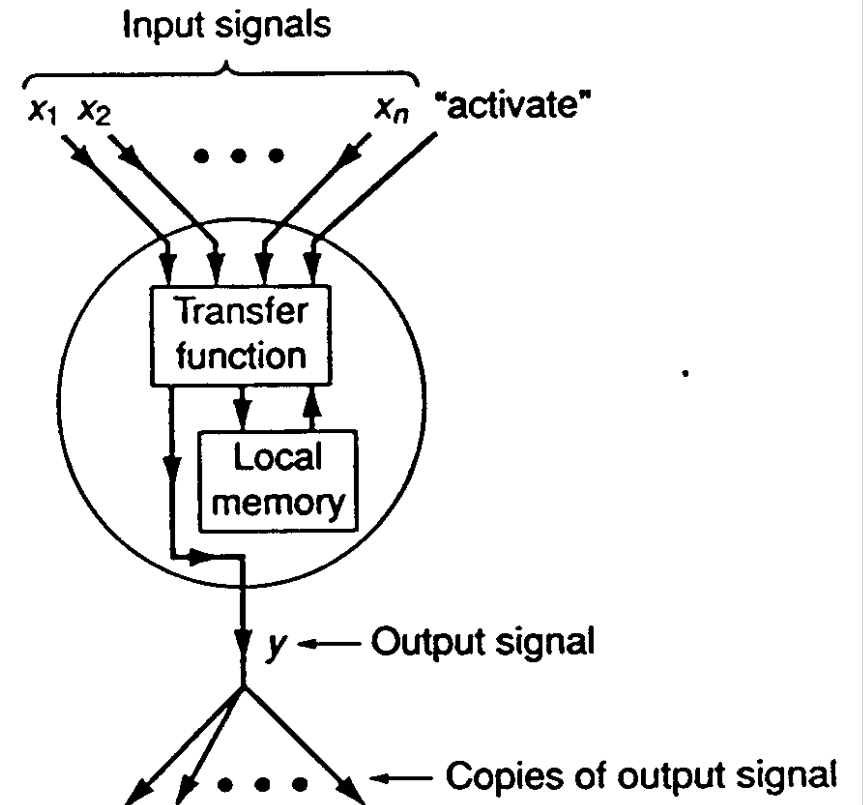
## **Una definizione formale di Rete Neurale (Hecht-Nielsen)**

Una *Rete Neurale* è una struttura parallela che processa informazioni distribuite. Tale rete consta di elementi di processazione (*PEs* o *neuroni*- che possono avere una memoria locale e che sono in grado di processare localmente informazioni) interconnessi tramite canali (detti *connessioni*) che trasmettono segnali unidirezionali. Ogni neurone ha una singola connessione di uscita che si dirama in un certo numero di connessioni collaterali; ognuna di questa trasporta lo stesso segnale - il segnale d'uscita del neurone. Questo segnale d'uscita può essere di qualunque tipo matematico. La computazione compiuta all'interno di ciascun neurone può essere definita arbitrariamente con l'unica restrizione che deve essere completamente locale; cioè deve dipendere solo dai valori correnti dei segnali d'ingresso che arrivano al neurone tramite opportune connessioni e dai valori immagazzinati nella memoria locale del neurone.

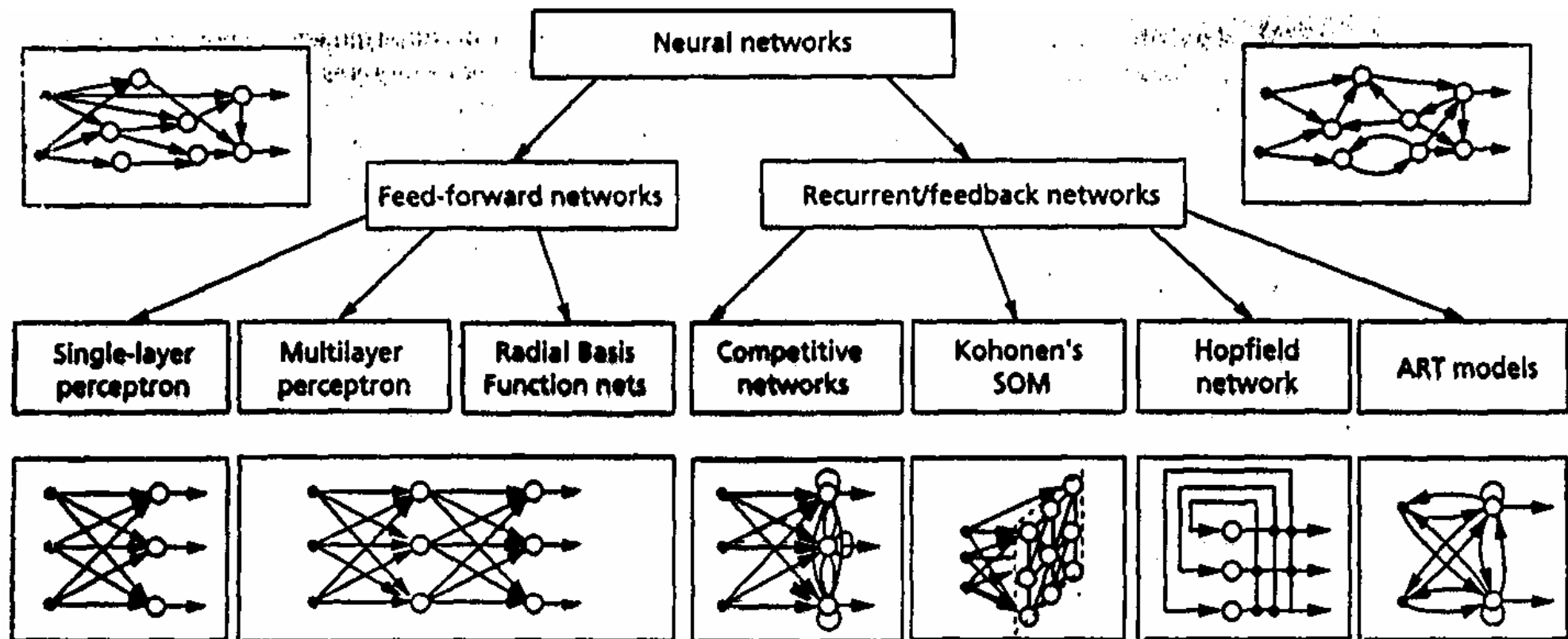
# Una possibile Architettura



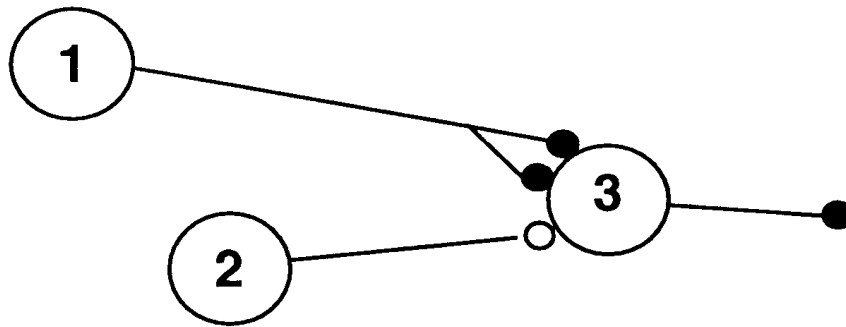
# Il generico neurone



# Una possibile Tassonomia (Jain 97)



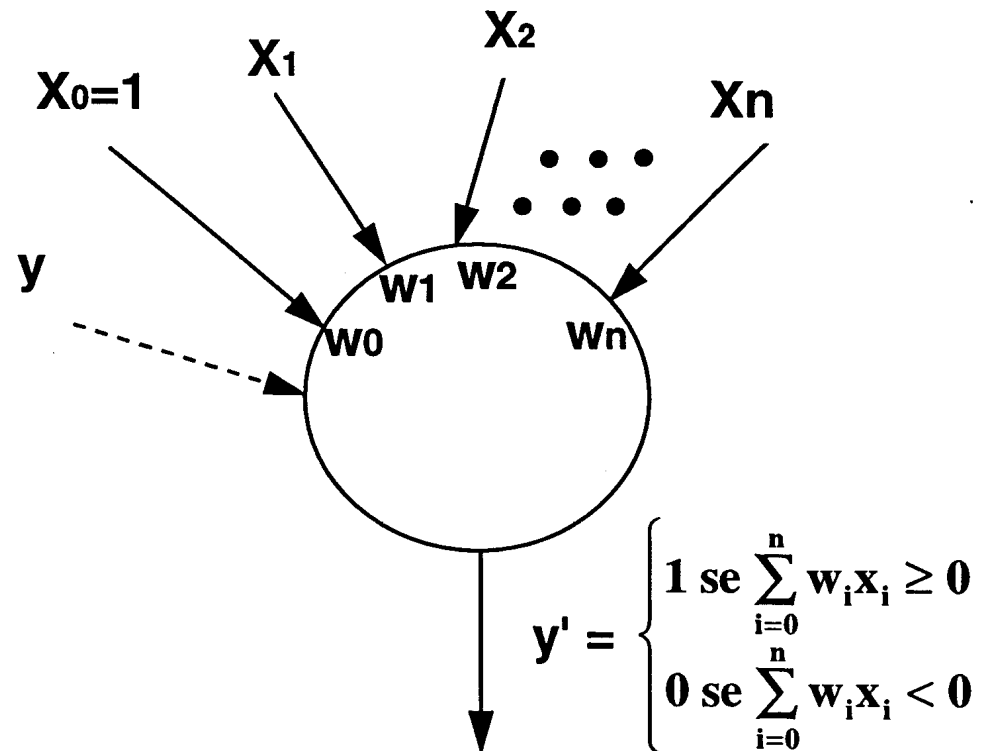




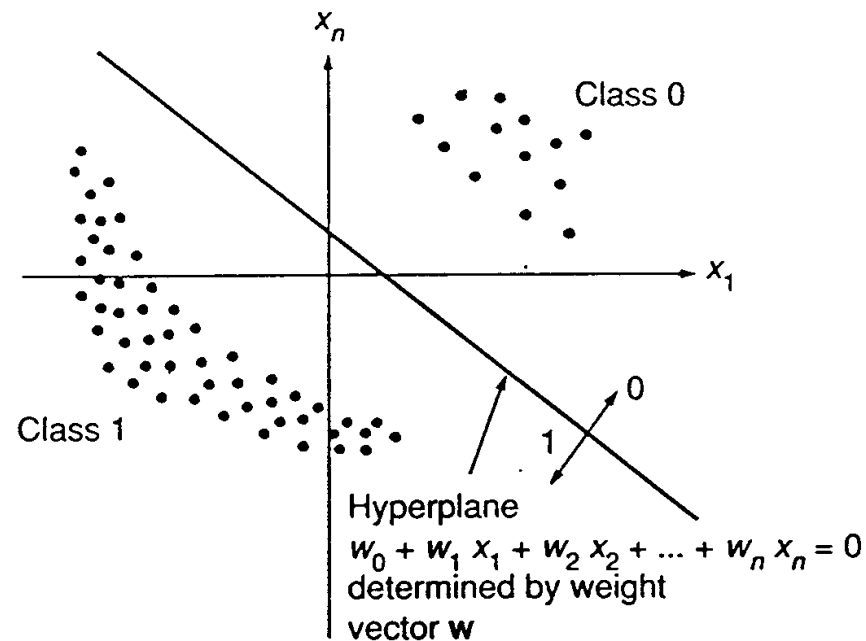
$$N3(t) = N1(t-1) \& (\text{not } N2(t-1))$$

## Il modello di McCulloch-Pitts (1943)

## Il Percettrone di Rosenblatt (1957)



# Gli Iperpiani



## Legge di apprendimento

$$W^{\text{new}} = W^{\text{old}} + (y - y') x$$

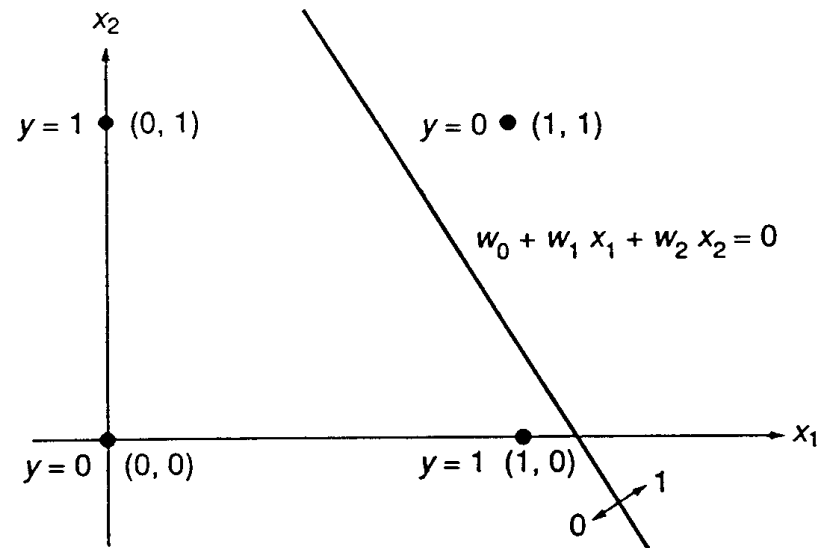
## Un problema semplice: la OR

$x_1$	$x_2$	$y$
0	0	0
0	1	1
1	0	1
1	1	1

## Condizione iniziale:

$$W^0 = [0.5, 0, 0]$$

# La critica di Minsky (1969): il caso della XOR

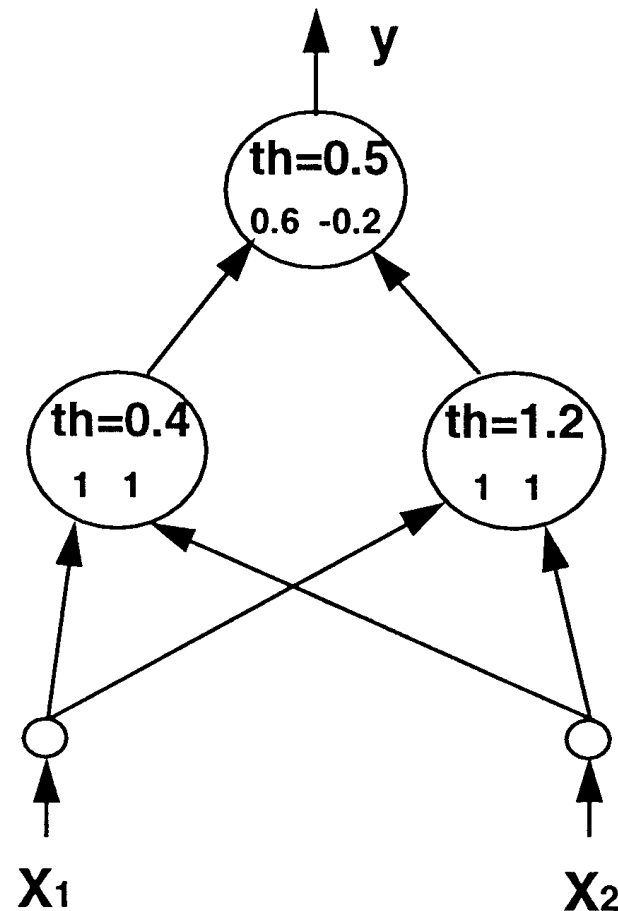


Impossibility of computing the EXCLUSIVE OR (XOR) function with a perceptron. In order for the perceptron to compute XOR, a line (the hyperplane corresponding to the weights  $w_0$ ,  $w_1$  and  $w_2$ ) must be found that puts points  $(0,0)$  and  $(1,1)$  on one side of the line and  $(0,1)$  and  $(1,0)$  on the other side of the line. This is obviously impossible. This trivial proof was generalized and expanded by Minsky and Papert in the mid- 1960's into a damnation of the entire field of neurocomputing.

## Problemi del Percettrone:

- Classificatore lineare
- Scaling

# Una possibile soluzione: il Percettrone Multilivello

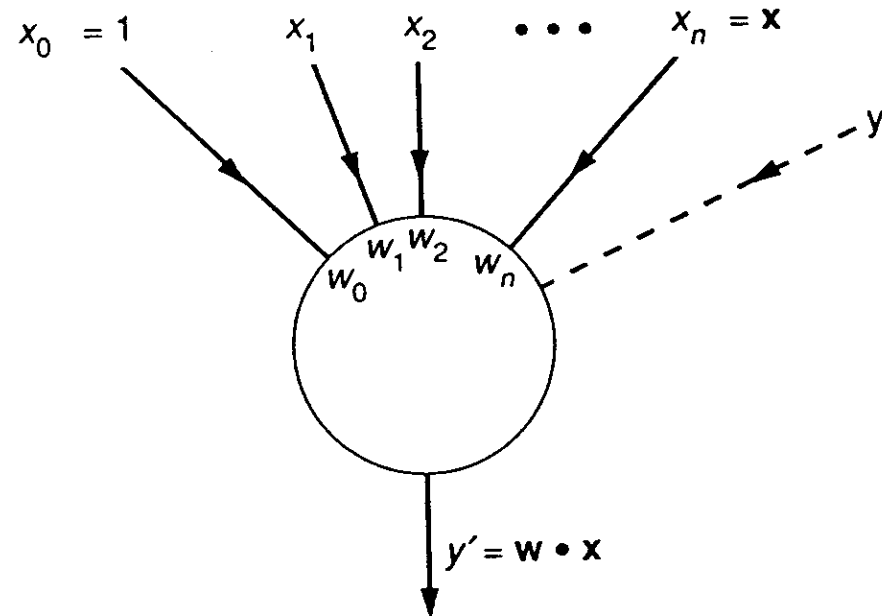


**La rinascita:**

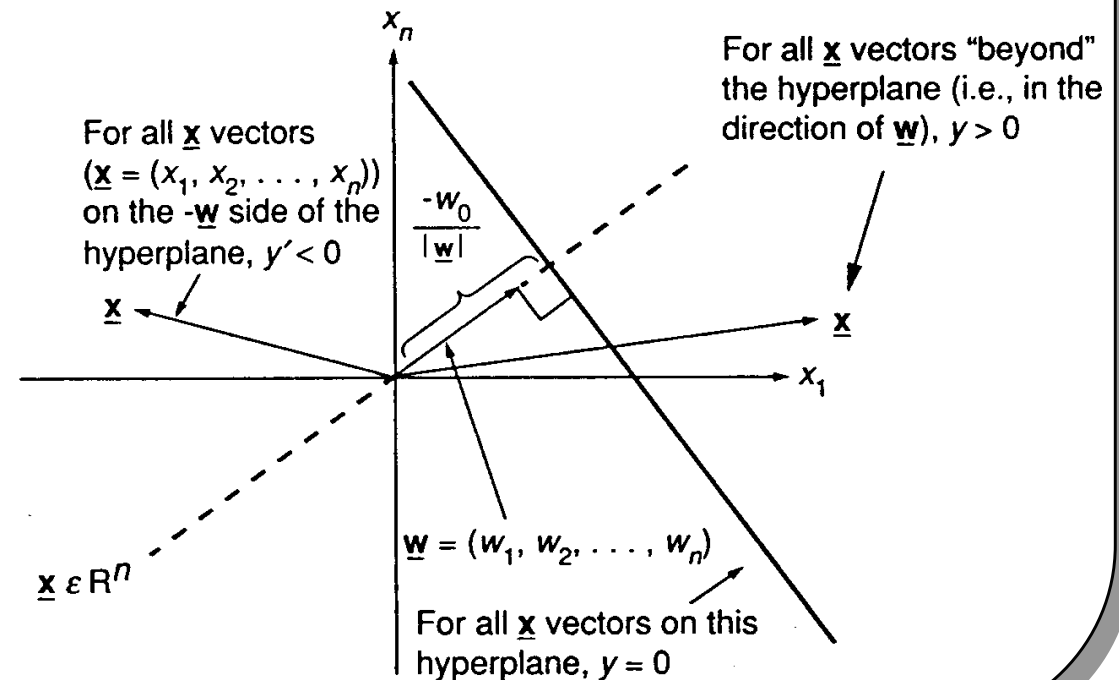
- I libri del PDP Group (1986)
- 1° IEEE ICNN (San Diego, 1987)
- IEEE Transactions on Neural Networks (1990)

# Adaline

The ADALINE (ADaptive LiNear Element), or *affine combiner*. The vector  $\mathbf{x} = (x_0, x_1, \dots, x_n)$  is entered into the processing element, and the number  $y' = \mathbf{w} \cdot \mathbf{x} = w_0x_0 + w_1x_1 + \dots + w_nx_n = w_0 + w_1x_1 + \dots + w_nx_n$  is emitted, where  $\mathbf{w} = (w_0, w_1, \dots, w_n)$  is the weight vector of the ADALINE processing element. Note that the zero<sup>th</sup> component of  $\mathbf{x}$  (namely,  $x_0$ ) is always equal to 1. This is called a *bias input*. The ADALINE uses Widrow learning.



The geometry of the ADALINE input/output relationship. If the vector  $\underline{\mathbf{x}} = (x_1, x_2, \dots, x_n)^T$  lies on the hyperplane perpendicular to  $\underline{\mathbf{w}} = (w_1, w_2, \dots, w_n)^T$  at directed distance  $(-w_0 / |\underline{\mathbf{w}}|)$  from the origin, then the output of the ADALINE will be 0. The  $\underline{\mathbf{x}}$  vectors lying on the "far side" of this hyperplane have positive  $y'$  output;  $\underline{\mathbf{x}}$  vectors lying on the  $-\underline{\mathbf{w}}$  side of the hyperplane have negative  $y'$  output. The output of the ADALINE depends linearly on the distance of  $\mathbf{x}$  from the  $\mathbf{w}$  hyperplane.



# L'errore quadratico medio

$$F(\mathbf{w}) = \lim_{N \rightarrow \infty} \frac{1}{N} \sum_{k=1}^N (y_k - y'_k)^2 = E[(y_k - y'_k)^2]$$

## Trovo il vettore dei pesi ottimo

$$\begin{aligned} F(\mathbf{w}) &= E[(y_k - \mathbf{w}^T \mathbf{x}_k)^2] = \\ &= E[(y_k^2 - 2y_k \mathbf{w}^T \mathbf{x}_k - \mathbf{w}^T \mathbf{x}_k \mathbf{x}_k^T \mathbf{w})] = \\ &= E[y_k^2] - 2\mathbf{w}^T E[y_k \mathbf{x}_k] + \mathbf{w}^T E[\mathbf{x}_k \mathbf{x}_k^T] \mathbf{w} = \\ &= \mathbf{p} - 2\mathbf{w}^T \mathbf{q} + \mathbf{w}^T \mathbf{R} \mathbf{w} \end{aligned}$$

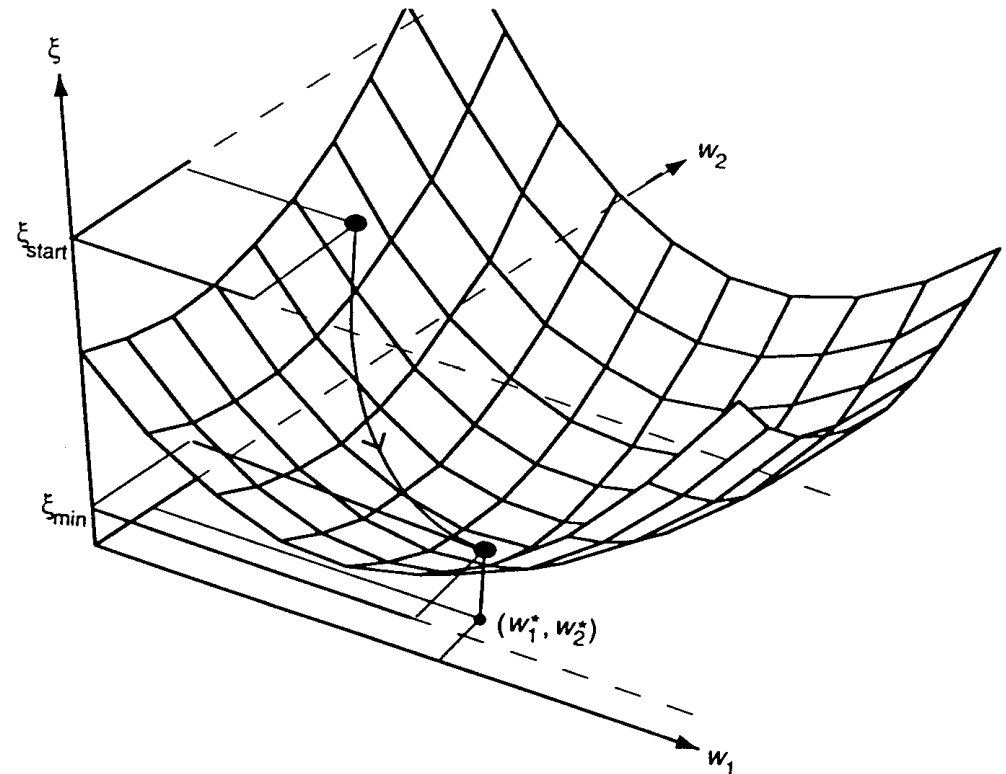
$$\begin{aligned} \nabla F(\mathbf{w}) &= \nabla(\mathbf{p} - 2\mathbf{w}^T \mathbf{q} + \mathbf{w}^T \mathbf{R} \mathbf{w}) = \\ &= -2\mathbf{q} + 2\mathbf{R} \mathbf{w} \end{aligned}$$

$$\nabla F(\mathbf{w}) = \mathbf{0} \Rightarrow \mathbf{w}^* = \mathbf{R}^{-1} \mathbf{q}$$

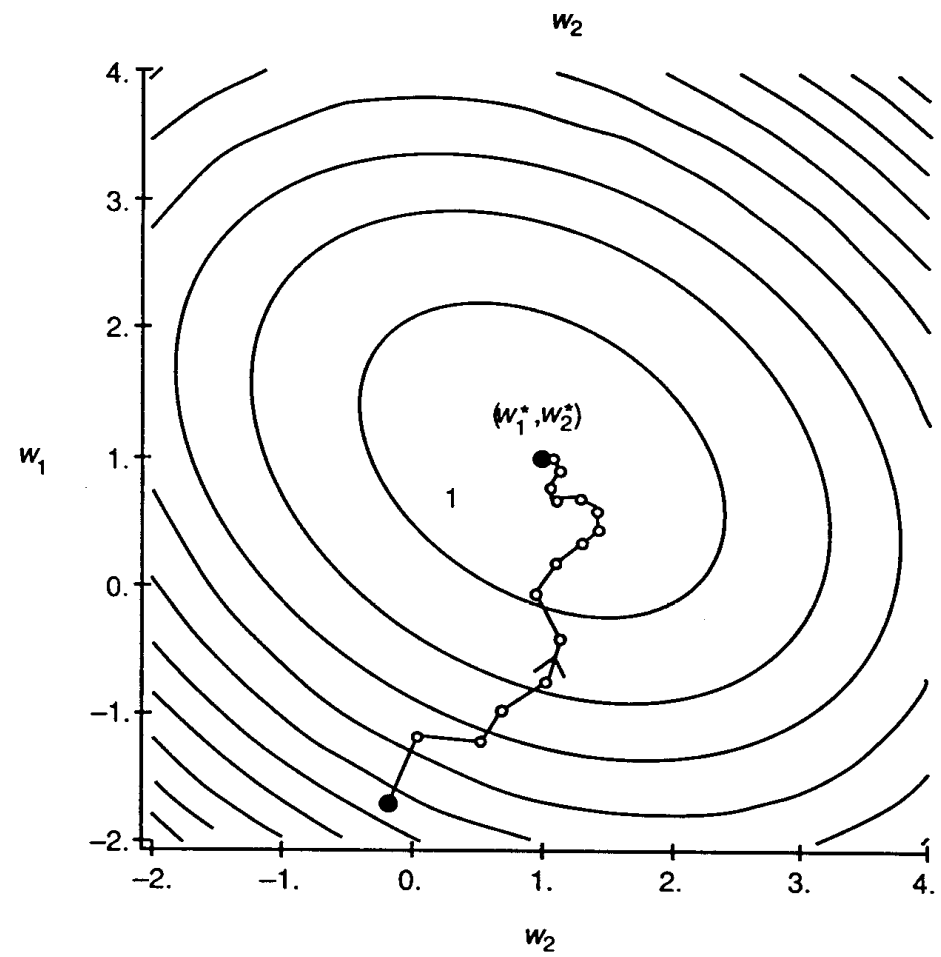
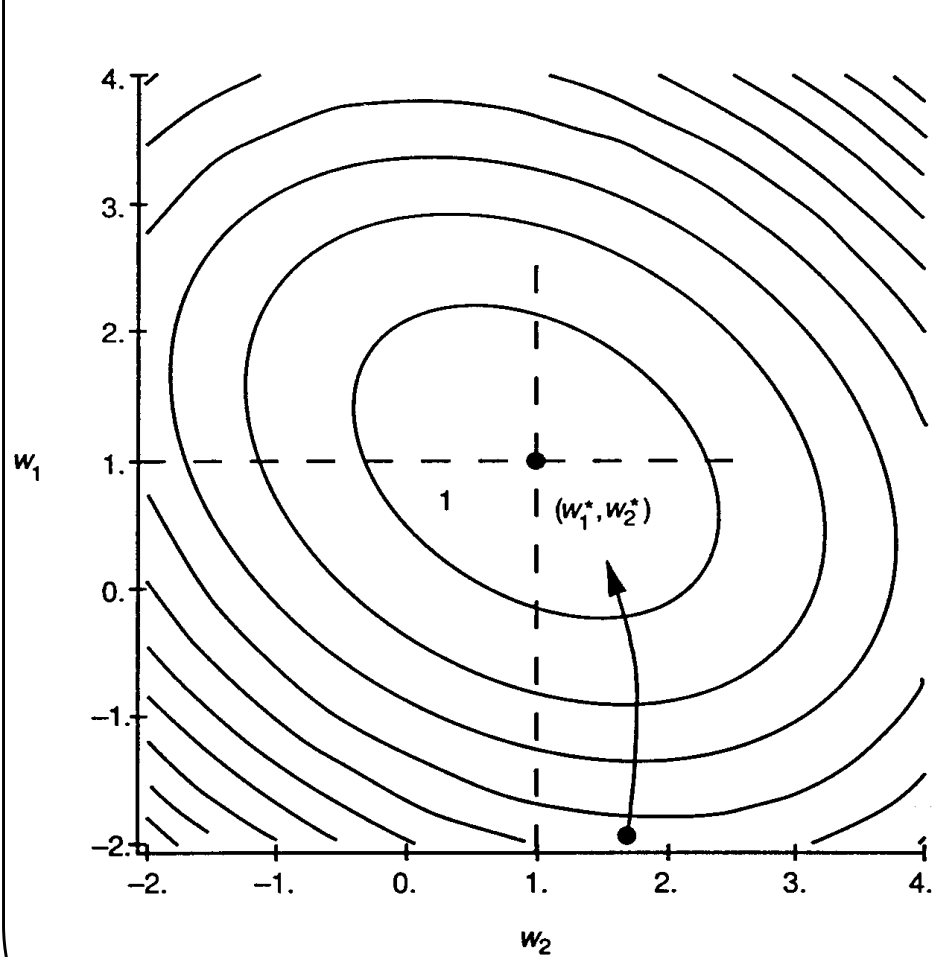
# La Legge di apprendimento di Widrow-Hoff (Delta Rule)

$$\begin{aligned}\nabla_{\mathbf{w}} F(\mathbf{w}) &= \nabla \left[ \lim_{N \rightarrow \infty} \frac{1}{N} \sum_{k=1}^N (y_k - y'_k)^2 \right] = \\ &= \lim_{N \rightarrow \infty} \frac{1}{N} \sum_{k=1}^N 2(y_k - y'_k) \nabla y'_k = \\ &= \lim_{N \rightarrow \infty} \frac{1}{N} \sum_{k=1}^N 2(y_k - y'_k)(-\mathbf{x}_k) = \\ &= -2E[(y_k - y'_k)\mathbf{x}_k]\end{aligned}$$

$$\Delta \mathbf{w}_k = \eta (y_k - y'_k) (\mathbf{x}_k)$$

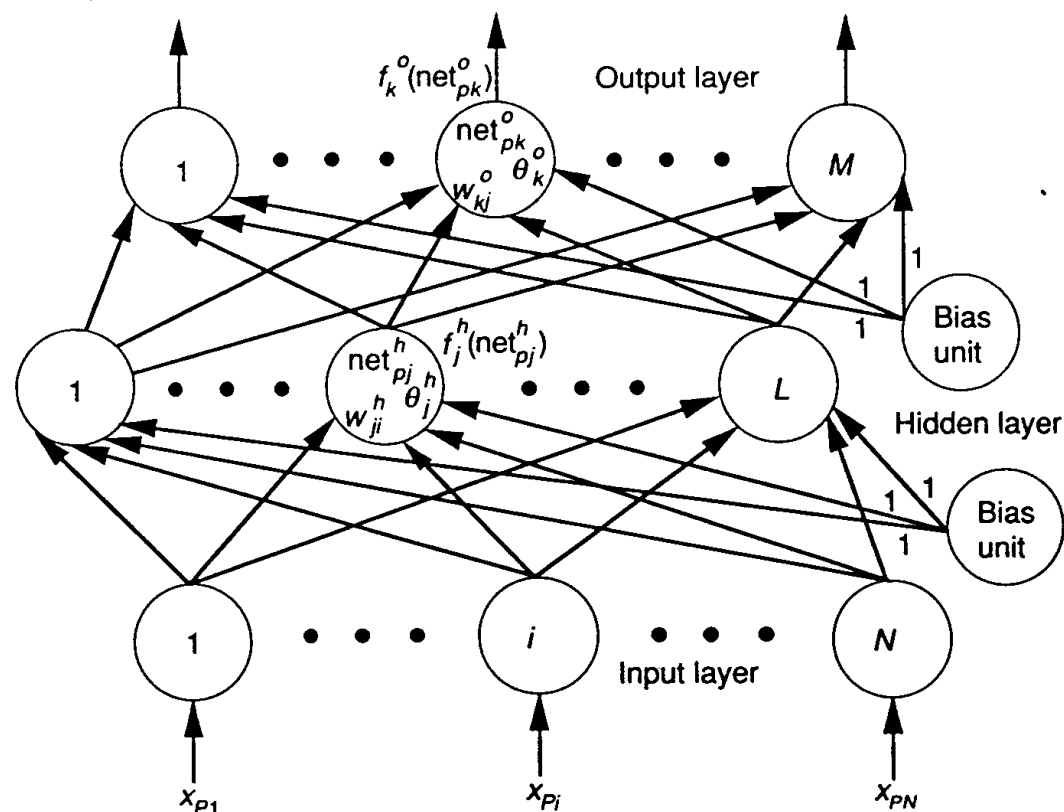


# La modifica dei pesi





# Il Percettrone Multilivello



$$net_{pj}^h = \sum_{i=1}^N w_{ji}^h x_{pi} + \theta_j^h$$

# input    # neuron

$$i_{pj} = f_j^h(net_{pj}^h)$$

$$net_{pk}^o = \sum_{j=1}^L w_{kj}^o i_{pj} + \theta_k^o$$

$$o_{pk} = f_k^o(net_{pk}^o)$$

# **L'Algoritmo Back-Propagation**

- Algoritmo di apprendimento supervisionato (Rumelhart, Hinton e Williams, 1986)

**PROCEDURE Back-Propagation learning ()**

**BEGIN**

**REPEAT**

**E = 0;**

**FOR r := 1 TO N<sub>tr</sub>**

**BEGIN**

**forward();**

**valutazione di E;**

**backward();**

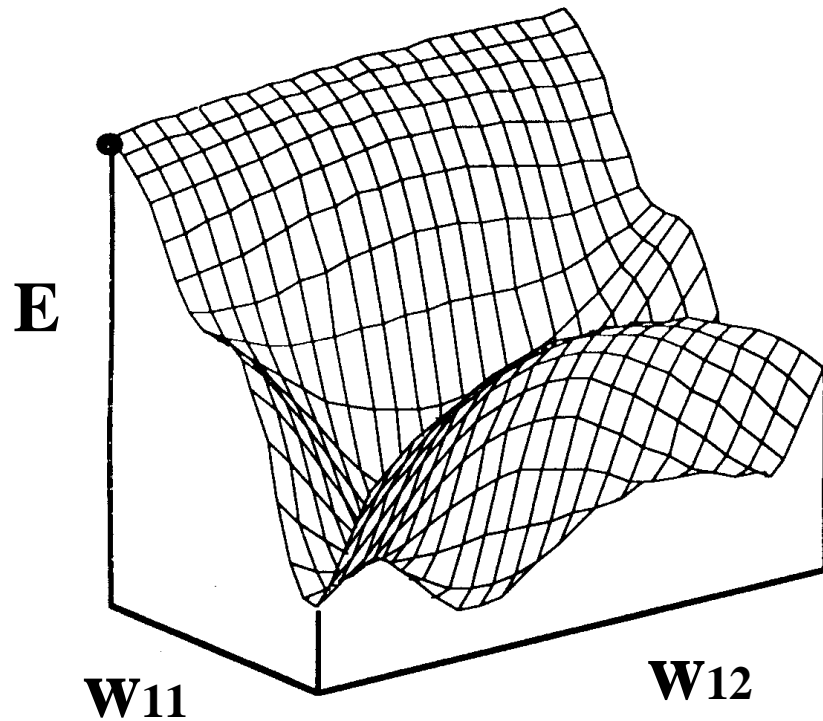
**END;**

**UNTIL (condizione di stop(E))**

**END;**

# Il Percettrone Multilivello: Addestramento

Superficie d'Errore



Aggiornamento dei pesi

$$E = \frac{1}{2} \sum_j (D_j - O_j)^2$$
$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$$

# La modifica dei pesi: Output-Hidden Layer

$$E_p = \frac{1}{2} \sum_{k=1}^M \delta_{pk}^2 = \frac{1}{2} \sum_{k=1}^M (y_{pk} - o_{pk})^2$$

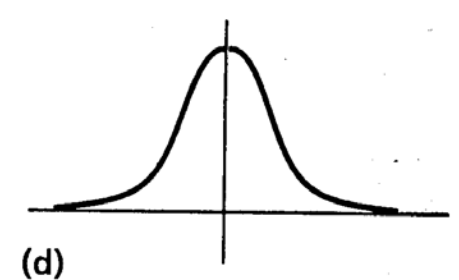
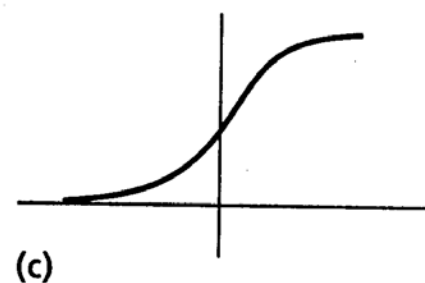
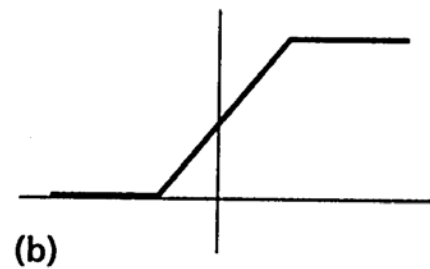
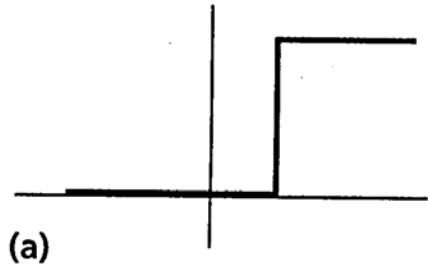
$$\frac{\partial E}{\partial w_{kj}^o} = -(y_{pk} - o_{pk}) \frac{\partial f_k^o}{\partial (\text{net}_{pk}^o)} \frac{\partial (\text{net}_{pk}^o)}{\partial w_{kj}^o}$$

$$\frac{\partial (\text{net}_{pk}^o)}{\partial w_{kj}^o} = \left( \frac{\partial}{\partial w_{kj}^o} \sum_{j=1}^L w_{kj}^o i_{pj} + \theta_k^o \right) = i_{pj}$$

$$-\frac{\partial E}{\partial w_{kj}^o} = (y_{pk} - o_{pk}) f_k^{o'}(\text{net}_{pk}^o) i_{pj}$$

$$\Delta_p w_{kj}^o = \eta (y_{pk} - o_{pk}) f_k^{o'}(\text{net}_{pk}^o) i_{pj}$$

# Funzioni di attivazione



(a) a gradino    (b) lineare a tratti    (c) sigmoidale    (d) gaussiana

$$f_k^o(\text{net}_{jk}^o) = \text{net}_{jk}^o$$

$$f_k^o(\text{net}_{jk}^o) = \frac{1}{1 + e^{-\text{net}_{jk}^o}} \Rightarrow f_k^{o'}(\text{net}_{jk}^o) = f_k^o(1 - f_k^o) = o_{jk}(1 - o_{jk})$$

$$\Delta_p w_{kj}^o = \eta(y_{pk} - o_{pk})o_{pk}(1 - o_{pk})i_{pj}$$

# La modifica dei pesi: Hidden-Input Layer

$$E^b = \frac{1}{2} \sum_k (y_{pk} - o_{pk})^2 = \frac{1}{2} \sum_k (y_{pk} - f_k^o(\sum_{j=1}^L w_{kj}^o i_{pj} + \theta_k^o))^2$$


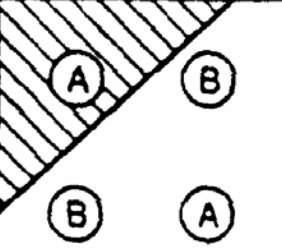
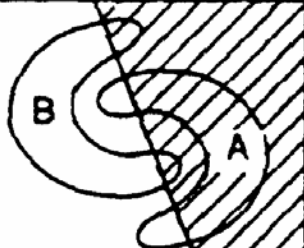
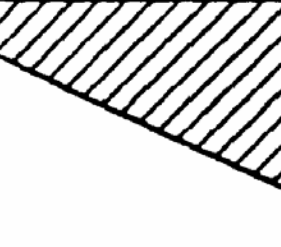
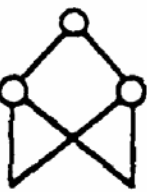
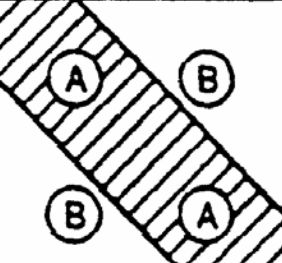
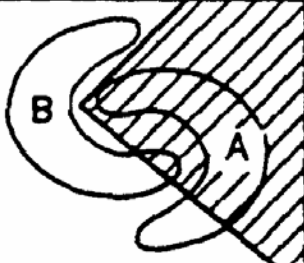
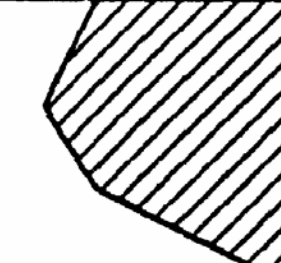
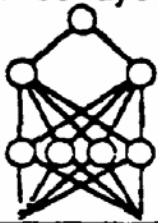
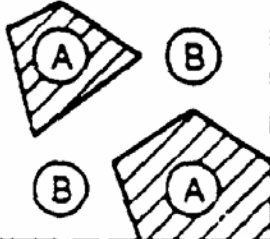
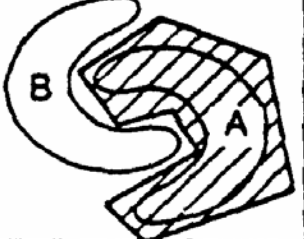
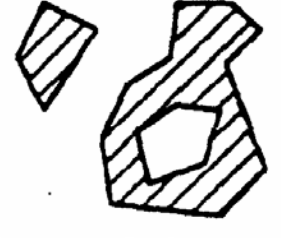
$$\frac{\partial E}{\partial w_{ji}^h} = \frac{1}{2} \sum_k \frac{\partial}{\partial w_{ji}^h} (y_{pk} - o_{pk})^2 =$$

$$= - \sum_k (y_{pk} - o_{pk}) \frac{\partial o_{pk}}{\partial (\text{net}_{pk}^o)} \frac{\partial (\text{net}_{pk}^o)}{\partial i_{pj}} \frac{\partial i_{pj}}{\partial (\text{net}_{pj}^h)} \frac{\partial (\text{net}_{pj}^h)}{\partial w_{ji}^h}$$

$$\frac{\partial E}{\partial w_{ji}^h} = - \sum_k (y_{pk} - o_{pk}) f_k^{o'}(\text{net}_{pk}^o) w_{kj}^o f_j^{h'}(\text{net}_{pj}^h) x_{pi}$$

$$\Delta_p w_{kj}^h = \eta f_j^{h'}(\text{net}_{pj}^h) x_{pi} \sum_k (y_{pk} - o_{pk}) f_k^{o'}(\text{net}_{pk}^o) w_{kj}^o$$

# Il Percettrone Multilivello : Regioni di decisione

Structure	Type of Decision Regions	Exclusive-OR Problem	Classes with Meshed Regions	Most General Region Shapes
<b>Single-layer</b> 	Half plane bounded by hyperplane			
<b>Two-layers</b> 	Convex open or closed regions			
<b>Three-layers</b> 	Arbitrary (Complexity limited by number of nodes)			

Lippman, 1987

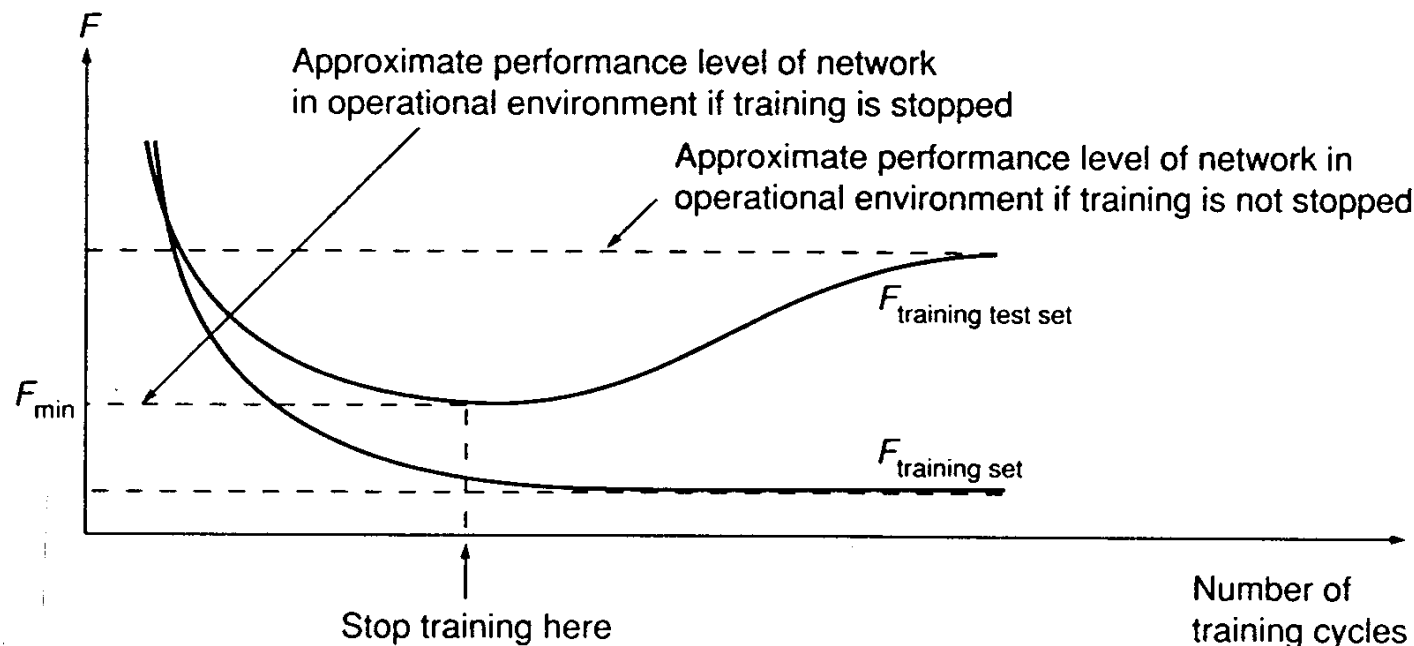
# Il Percettrone Multilivello: Addestramento

## Varianti:

- Batching
- Smoothing

## Problemi:

- Minimi locali
- Flat spots
- Overtraining

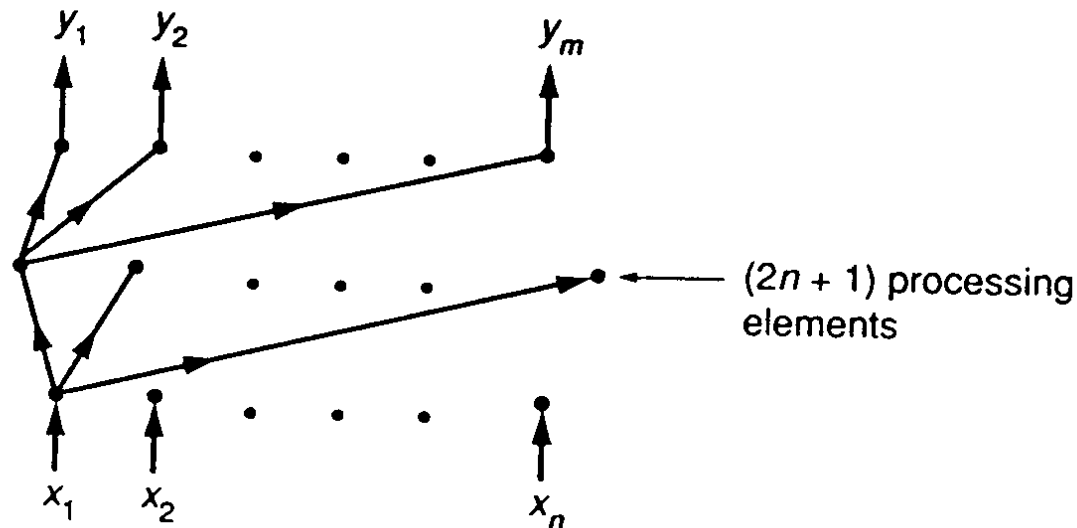


Training set error vs. training-test set error as a function of the number of training cycles. This illustrates the phenomenon of *overtraining* seen in some feature-based mapping networks.



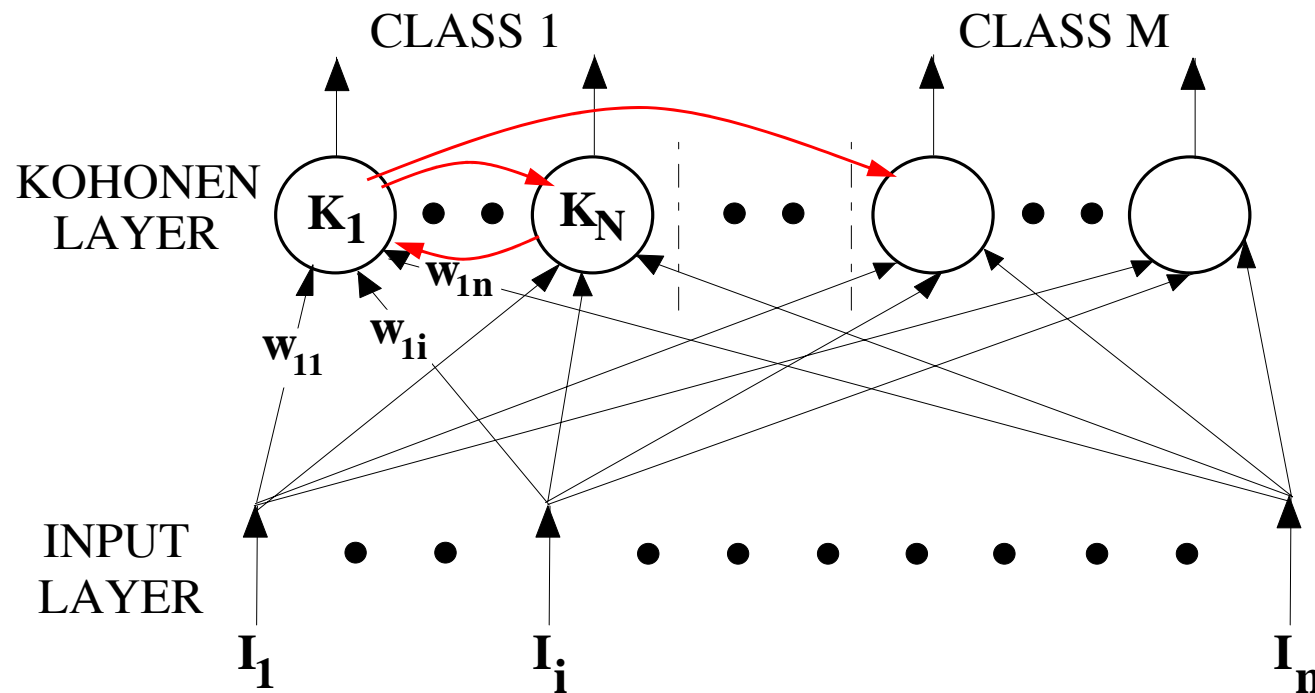
# Il Teorema di Kolmogorov

**Kolmogorov's Mapping Neural Network Existence Theorem:** *Given any continuous function  $f:[0,1]^n \rightarrow \mathbf{R}^m$ ,  $f(\mathbf{x}) = \mathbf{y}$ ,  $f$  can be implemented exactly by a three-layer feedforward neural network having  $n$  fanout processing elements in the first ( $\mathbf{x}$  - input) layer,  $(2n+1)$  processing elements in the middle layer, and  $m$  processing elements in the top ( $\mathbf{y}$  - output) layer*



- Topology of the Kolmogorov network. This network can *exactly* implement any continuous mapping. The network has three layers. The first layer consists of  $n$  input fanout units. The second layer consists of  $2n + 1$  semilinear units (i.e., the transfer function of these units is similar to a linear weighted sum). Finally, the third (output) layer has  $m$  processing elements with highly nonlinear transfer functions.

# PARADIGMA LVQ: Architettura



- Architettura a due strati **con connessioni laterali**
- E' un classificatore

# PARADIGMA LVQ: Addestramento

- Algoritmo di apprendimento: LVQ1 (Kohonen, 1990)

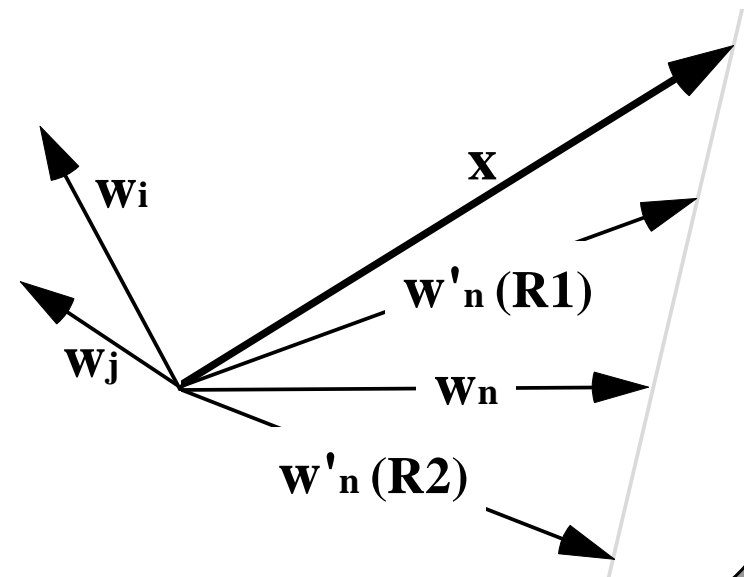
```
PROCEDURE Competitive Learning( )  
BEGIN  
  FOR i := 1 TO N*M DO  
    di = distanza(x, wi) ;  
  n = minimo(di) ;  
  IF classe[n] = classe[x]  
    THEN aggiorna con la regola R1  
    ELSE aggiorna con la regola R2  
END
```

Se considero la distanza euclidea...

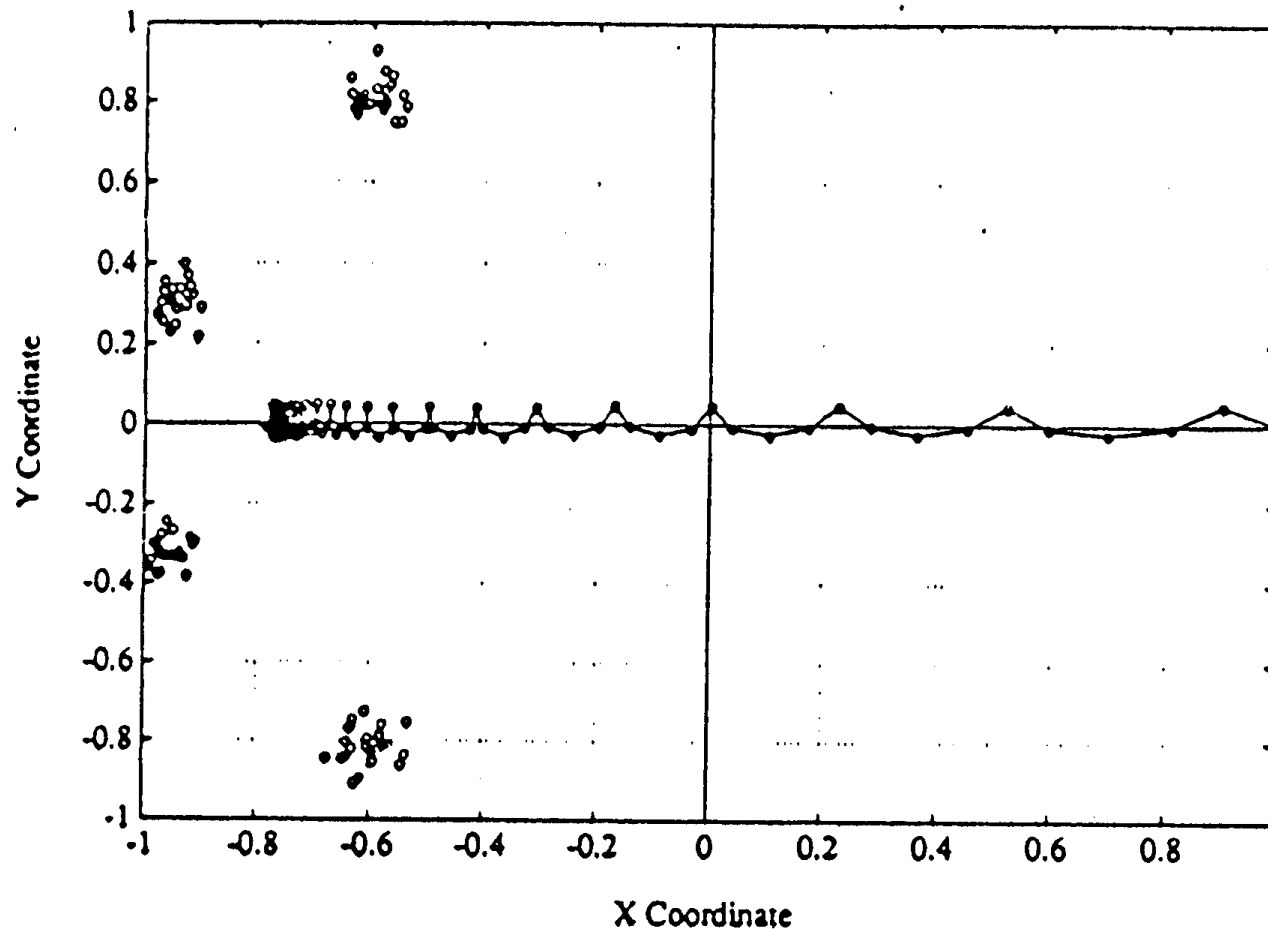
$$d_i = \|w_i - x\| = \sqrt{\sum_{j=1}^n (w_{ij} - x_j)^2}$$

$$R1: \Delta w_n = \alpha (x - w_n)$$

$$R2: \Delta w_n = -\gamma (x - w_n)$$



# PARADIGMA LVQ: Addestramento



Il problema del sottoutilizzo dei neuroni

## **PARADIGMA LVQ: Addestramento**

- Algoritmi di apprendimento alternativi
  - Coscienza (DeSieno, 1988)
  - LVQ2 (Kohonen, 1990)
  - FSCL (Ahalt, Krishnamurthy, Chen e Melton, 1990)
  - RPCL (Xu, Krzyzak e Oja, 1993)

# Due possibili soluzioni

## - FSCL

(Frequency Sensitive Competitive Learning)

## - C<sup>2</sup>L

(Algoritmo della Coscienza)

- Se l'algoritmo è supervisionato, c'è un'ulteriore competizione tra i neuroni che appartengono alla classe del vettore d'ingresso e si usa una distanza modificata per calcolare il *class winner*
- Altrimenti la distanza modificata è direttamente usata per calcolare il *network winner*

$$d'_i = d_i \cdot F_i$$

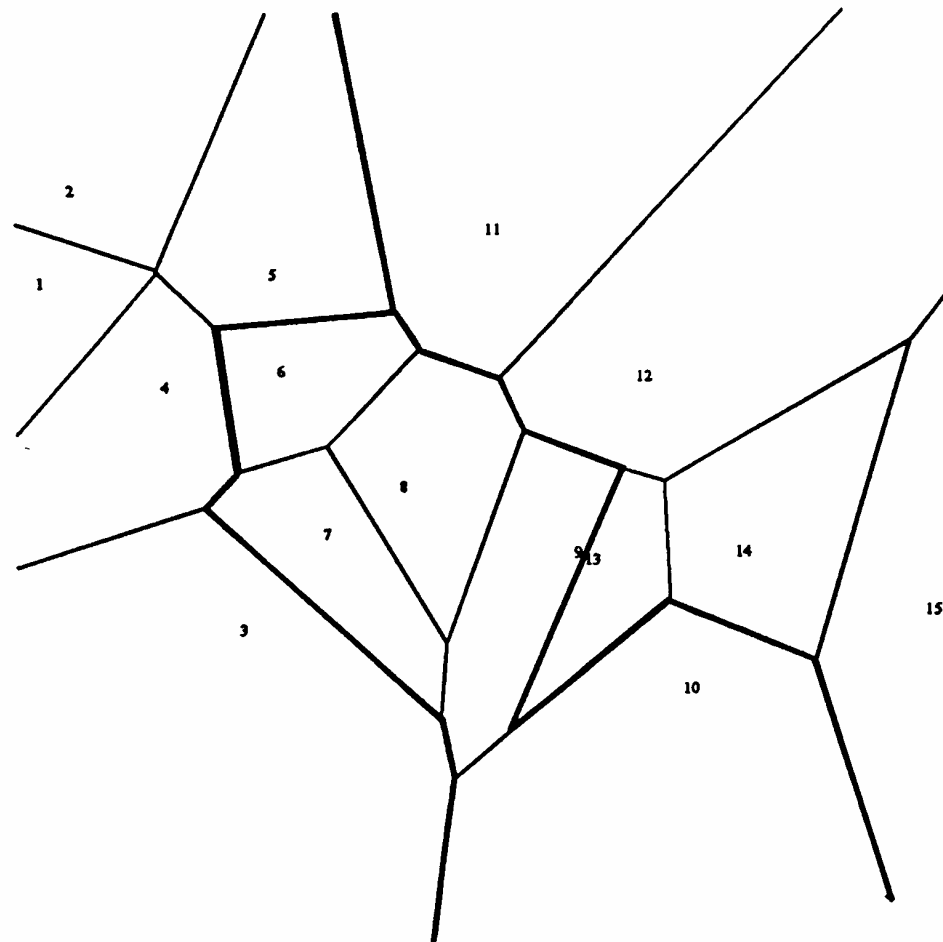
$$d''_i = d_i \cdot b_i$$

$$b_i = \frac{1}{\sum_{j=1}^N d_j}$$

- Regole di aggiornamento dei pesi (caso supervisionato)

$$R1 \begin{bmatrix} \Delta w_n \\ \Delta w_c \end{bmatrix} = \begin{bmatrix} 0 \\ \alpha (w_c - x) \end{bmatrix} \quad R2 \begin{bmatrix} \Delta w_n \\ \Delta w_c \end{bmatrix} = \begin{bmatrix} \gamma (w_n - x) \\ \beta (w_c - x) \end{bmatrix}$$

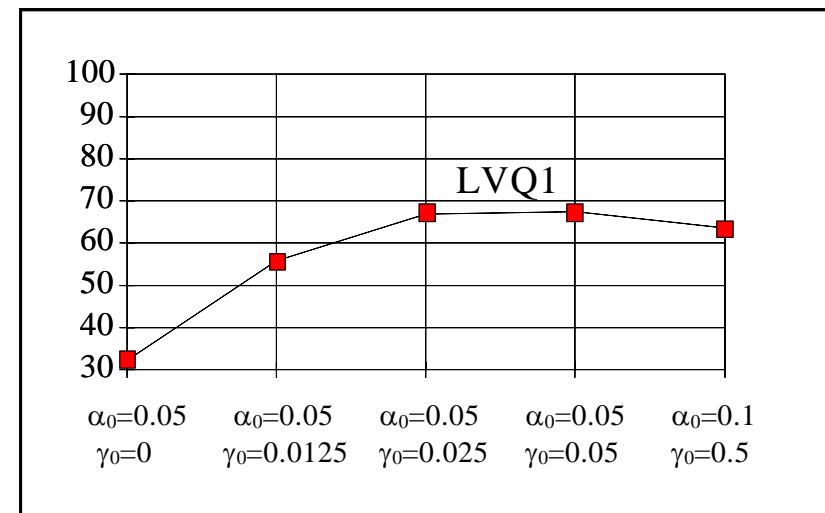
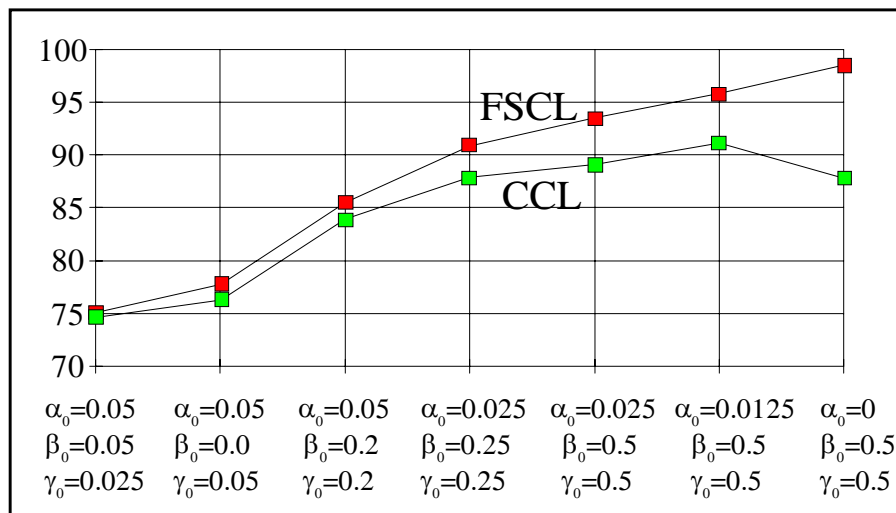
# PARADIGMA LVQ: Regioni di decisione



Tassellazione di Voronoi

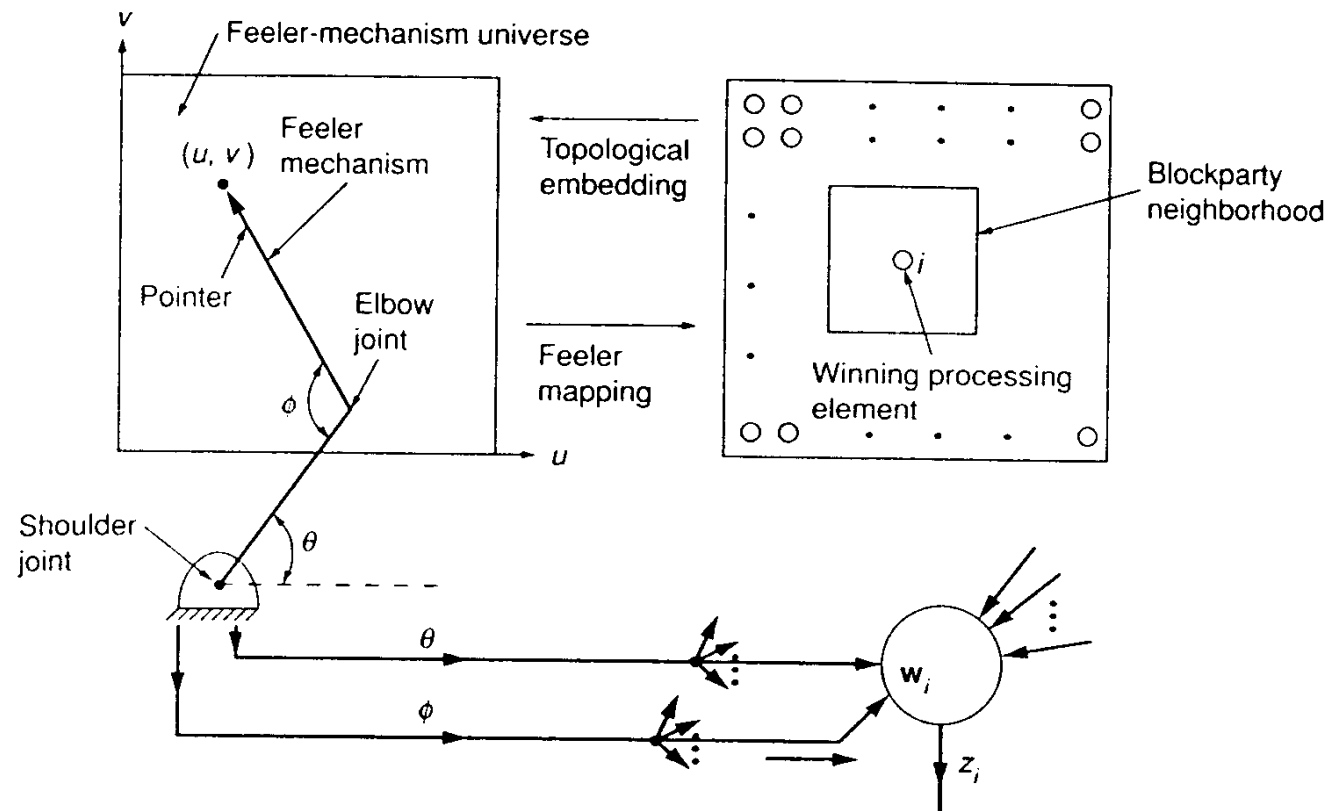
# PARADIGMA LVQ: Problemi e scelta dei parametri

- Legge di variazione dei coefficienti di apprendimento
- Valore iniziale dei coefficienti di apprendimento
- Valore di N





# Le Mappe Auto-Organizzanti (SOM) di Kohonen



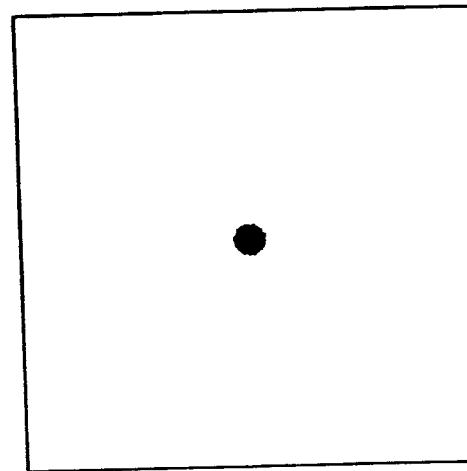
Self-organizing map training process. A point  $(u, v)$  is chosen at random in the feeler mechanism universe. The pointer is then moved to this point - sending signals  $\theta$  and  $\phi$  to the network (which then adapts to these values). This process continues through many thousands of trials.

**La legge di apprendimento:**

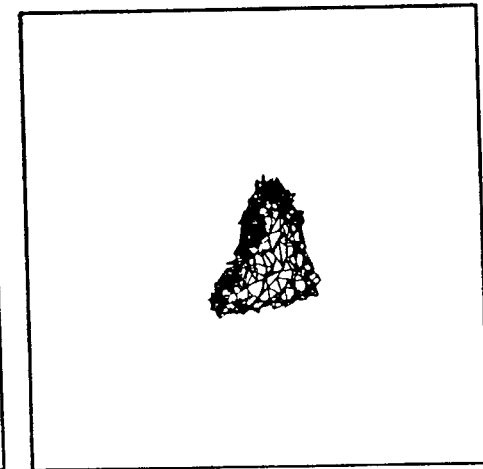
$$\mathbf{w}_i^{\text{new}} = \alpha_i(\mathbf{z}, t)(\theta, \phi) + [1 - \alpha_i(\mathbf{z}, t)]\mathbf{w}_i^{\text{old}}$$

# Esempi di funzionamento delle SOM

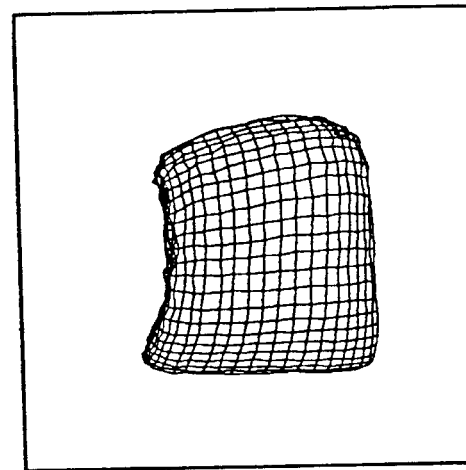
Four frames from a  
Kohonen self-organizing  
map neural network movie



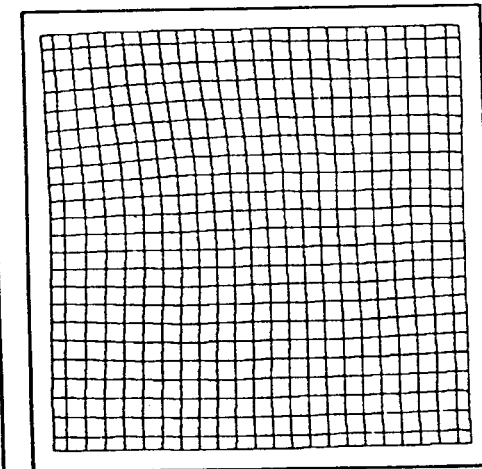
0



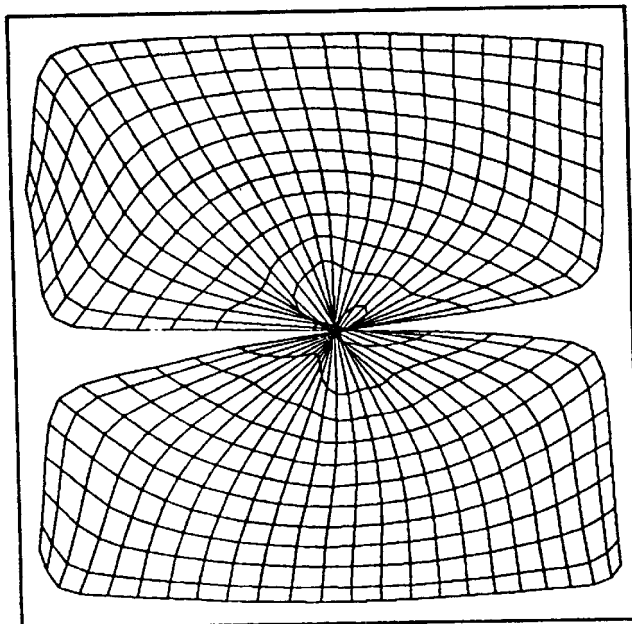
20



100



100000

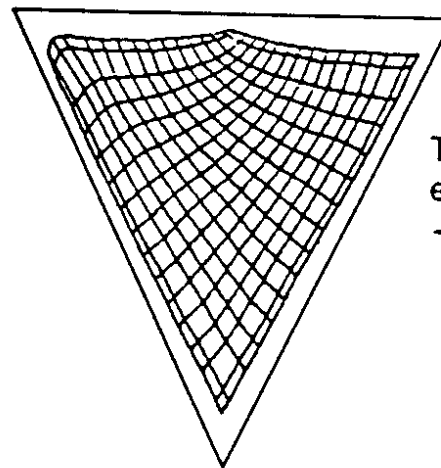


20000

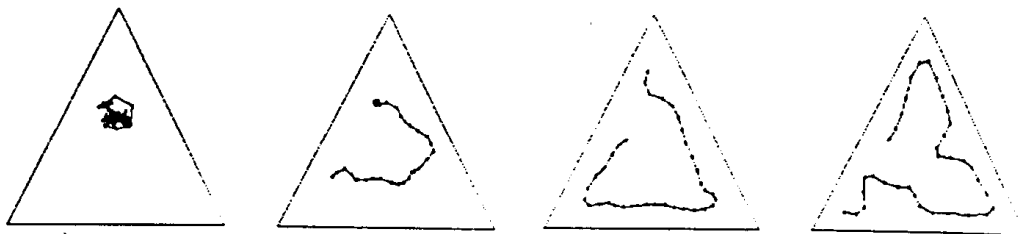
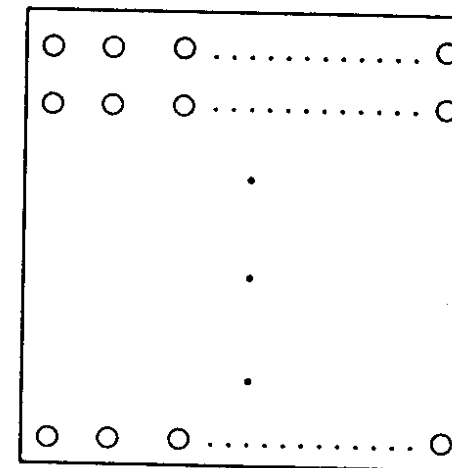
Example of a *twisted mesh*

# Esempi di funzionamento delle SOM

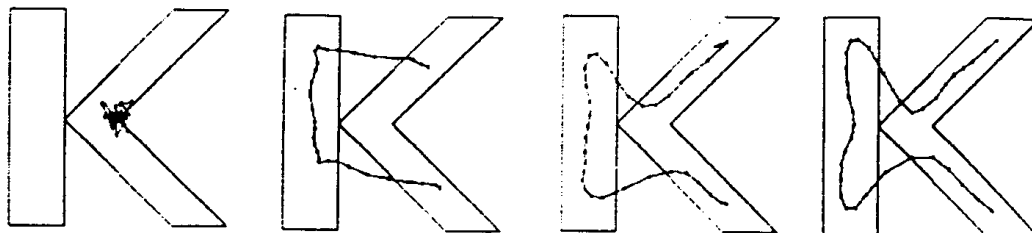
Kohonen movie frame showing the induced mapping from a rectangular neural network array to a triangular region of  $R^2$ .



Topological  
embedding



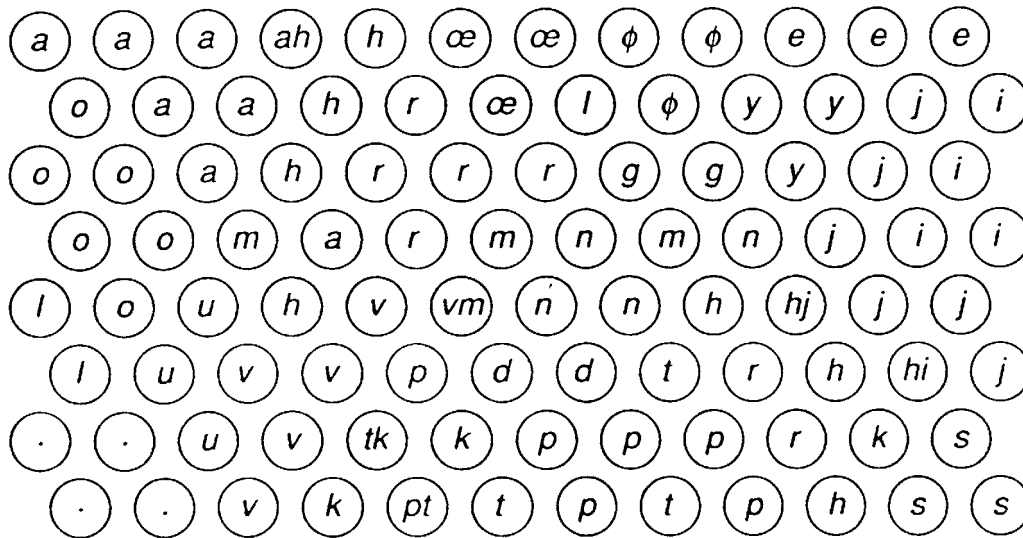
(a)



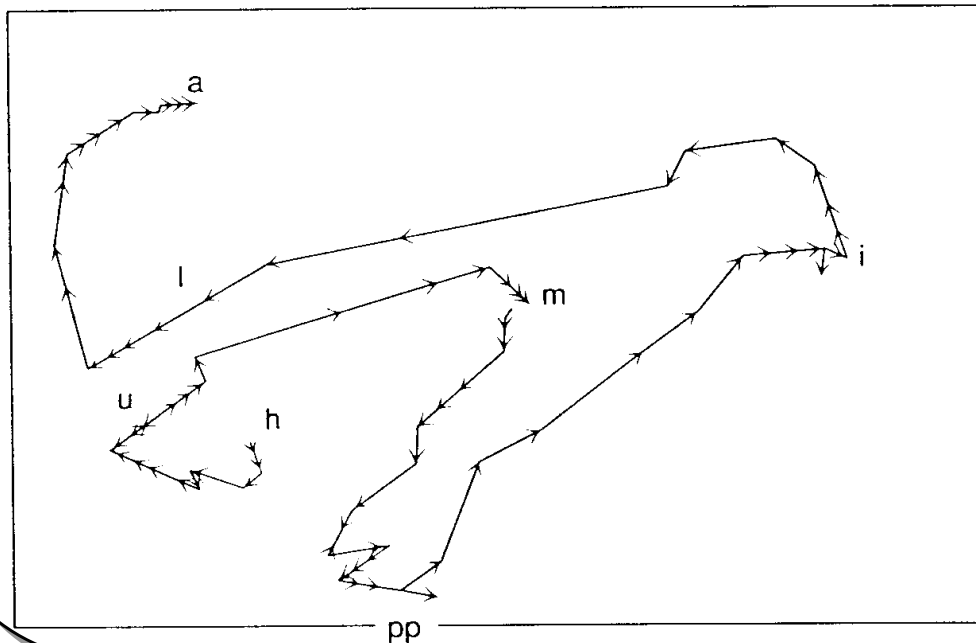
(b)

**SOM lineari**

# Neural Phonetic Typewriter (Kohonen, 1988)



This figure is a phonotopic map with the neurons, shown as circles, and the phonemes to which they learned to respond. A double label means that the node responds to more than one phoneme. Some phonemes -such as the plosives represented by *k*, *p*, and *t* - are difficult for the network to distinguish and are most prone to misclassification by the network.



This illustration shows the sequence of responses from the phonotopic map resulting from the spoken Finnish word *humppila*

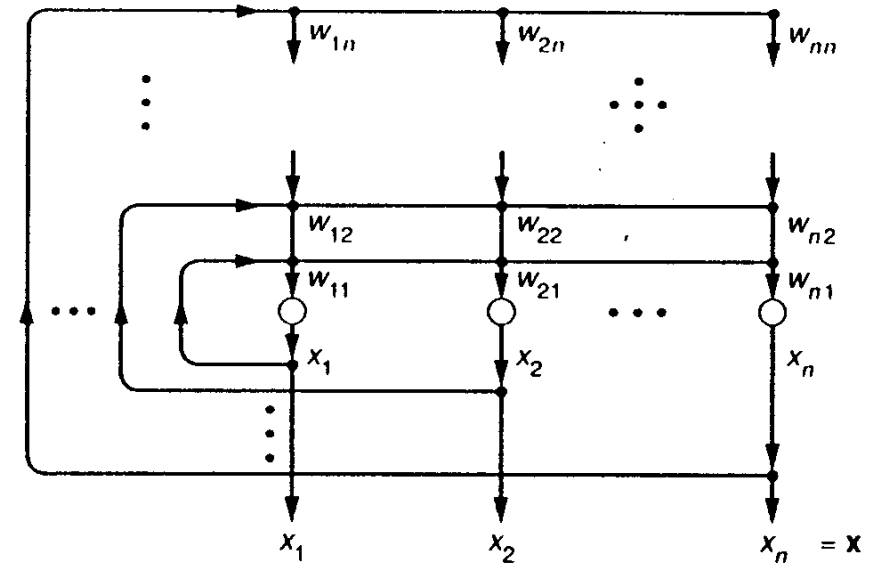
**Sampling time: 9.83 ms**

**Vettore di 15 componenti**

**Recognition rate: 92-97%**

# La rete di Hopfield

The Hopfield neural network. This recurrent associative network has  $n$  processing elements, each of which receives inputs from all the others. The input that a processing element receives from itself is ignored. All of the processing element output signals are bipolar. The network has an *energy function* associated with it; whenever a processing element changes state, this energy function always decreases. Starting at some initial position, the system's state vector simply moves downhill on the network's *energy surface* until it reaches a local minimum of the energy function. This convergence process is guaranteed to be completed in a fixed number of steps.



## La funzione di trasferimento:

$$\mathbf{x}_i^{\text{new}} = \begin{cases} 1 & \text{if } \sum_j \mathbf{w}_{ij} \mathbf{x}_j^{\text{old}} > T_i \\ \mathbf{x}_i^{\text{old}} & \text{if } \sum_j \mathbf{w}_{ij} \mathbf{x}_j^{\text{old}} = T_i \\ -1 & \text{if } \sum_j \mathbf{w}_{ij} \mathbf{x}_j^{\text{old}} < T_i \end{cases}$$

# La funzione di energia

$$H(\mathbf{x}) = -\sum_{i=1}^n \sum_{j=1}^n w_{ij} x_i x_j + 2 \sum_{i=1}^n T_i x_i$$

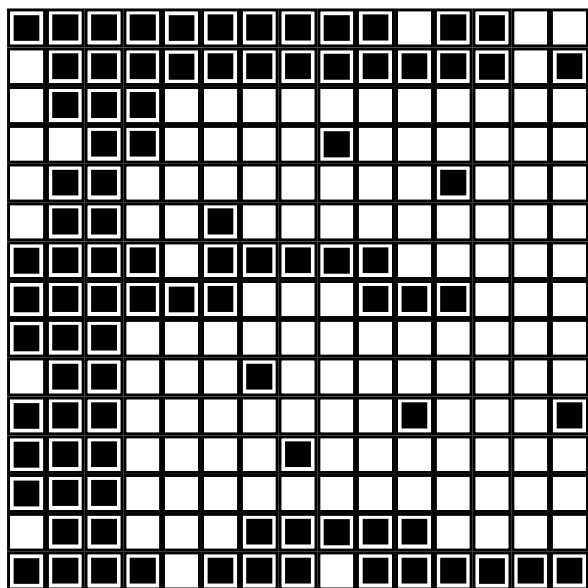
Calcolo la variazione:

$$\Delta H = H(\mathbf{x}^{\text{new}}) - H(\mathbf{x}^{\text{old}}) =$$

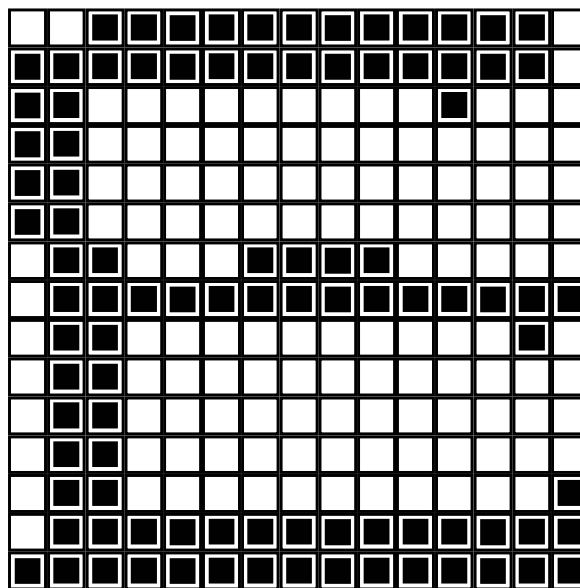
$$\begin{aligned} & -\sum_{i=1}^n \sum_{j=1}^n w_{ij} x_i^{\text{new}} x_j^{\text{new}} + 2 \sum_{i=1}^n T_i x_i^{\text{new}} + \sum_{i=1}^n \sum_{j=1}^n w_{ij} x_i^{\text{old}} x_j^{\text{old}} - 2 \sum_{i=1}^n T_i x_i^{\text{old}} = \\ & = -2 x_k^{\text{new}} \sum_{j=1}^n w_{kj} x_j^{\text{new}} + 2 T_k x_k^{\text{new}} + 2 x_k^{\text{old}} \sum_{j=1}^n w_{kj} x_j^{\text{old}} - 2 T_k x_k^{\text{old}} \end{aligned}$$

$$\Delta H = 2(x_k^{\text{old}} - x_k^{\text{new}}) \left[ \sum_{j=1}^n w_{kj} x_j^{\text{old}} - T_k \right] \leq 0$$

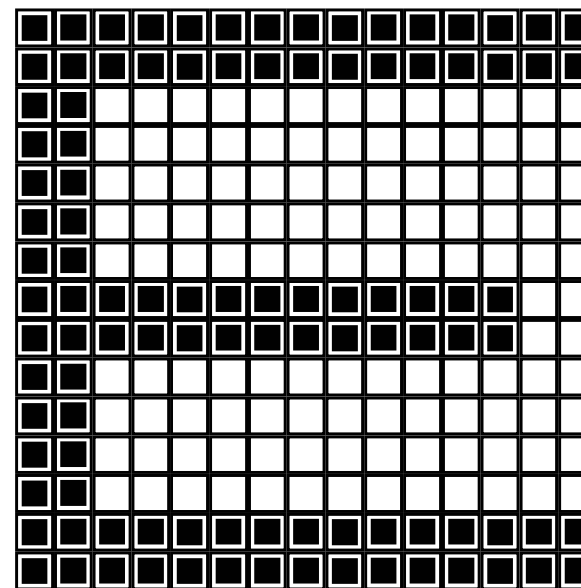
# Rete di Hopfield



Stato iniziale



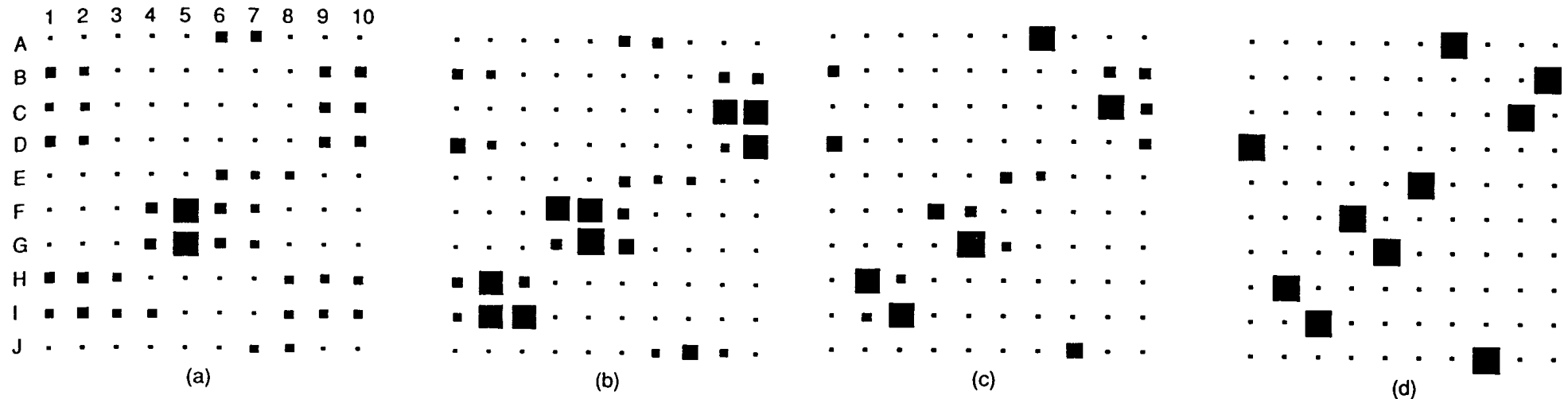
Dopo 2 iterazioni



Stato finale

# La rete di Hopfield continua

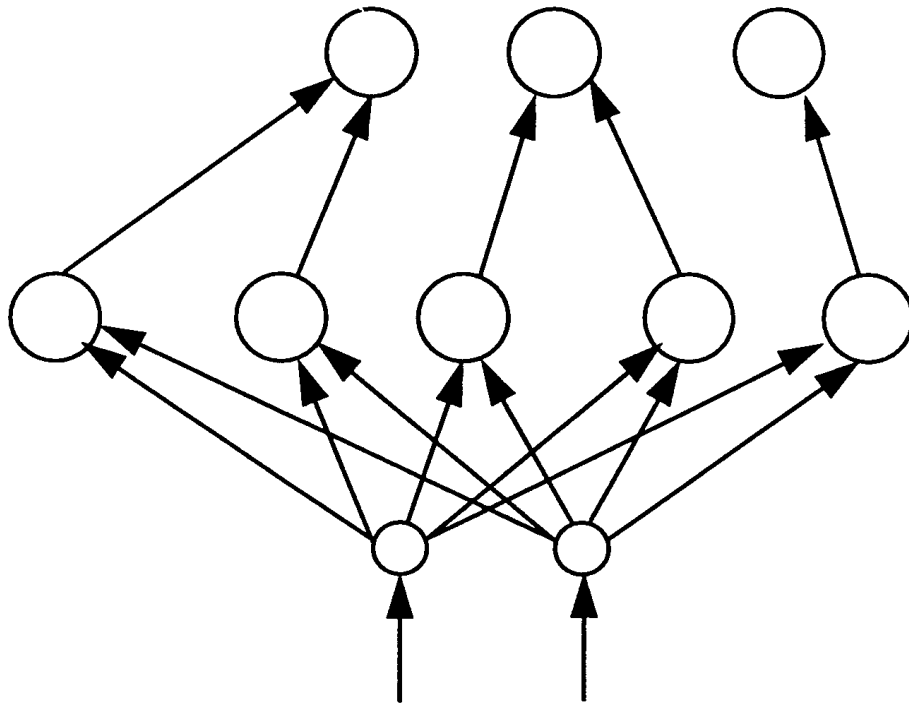
## Un applicazione: il TSP problem



This sequence of diagrams illustrates the convergence of the Hopfield network for a 10-city TSP tour. The output values,  $vX_I$ , are represented as squares at each location in the output unit matrix. The size of the square is proportional to the magnitude of the output value. (a, b, c) At the intermediate steps, the system has not yet settled on a valid tour. The magnitude of the output values for these intermediate steps can be thought of as the current estimate of the confidence that a particular city will end up in a particular position on the tour. (d) The network has stabilized on the valid tour, DHIFGEAJCB. *Source: Reprinted with permission of Springer-Verlag, Heidelberg, from J. J. Hopfield and D. W Tank, "Neural computation of decisions in optimization problems." Biological Cybernetics, 52:141-152, 1985.*



# Probabilistic Neural Network



$$\text{Out}_c = P_c \cdot \text{payoff}_c \cdot f_c(d_k)$$

$$f_c(d_k) = \frac{1}{m_c} \sum_{i=1}^{m_c} g(d_k)$$

$$g(d_k) = \frac{1}{(2\pi)^{n/2} \sigma^n} \exp\left(-\frac{d_k^2}{2\sigma^2}\right)$$

$$d_k = \|\mathbf{x} - \mathbf{w}_k\| = \sqrt{\sum_{i=1}^n (x_i - w_{ki})^2}$$

# Classificatore Bayesiano

$$p(C_i|X) = \frac{p(X|C_i)p(C_i)}{p(X)}$$

- Massimizza la probabilità a posteriori
- Classificatore ottimo

Problema: in genere le distribuzioni non sono note

## **Una possibile Tassonomia (Lippman 93)**

- **Classificatori basati sulla stima della pdf**
  - *parametrici*
- **Classificatori basati sulla stima della prob. a posteriori**
  - stime locali: PNN
  - stime globali: MLP
- **Classificatori che tracciano le regioni di decisione**
  - LVQ, Hopfield