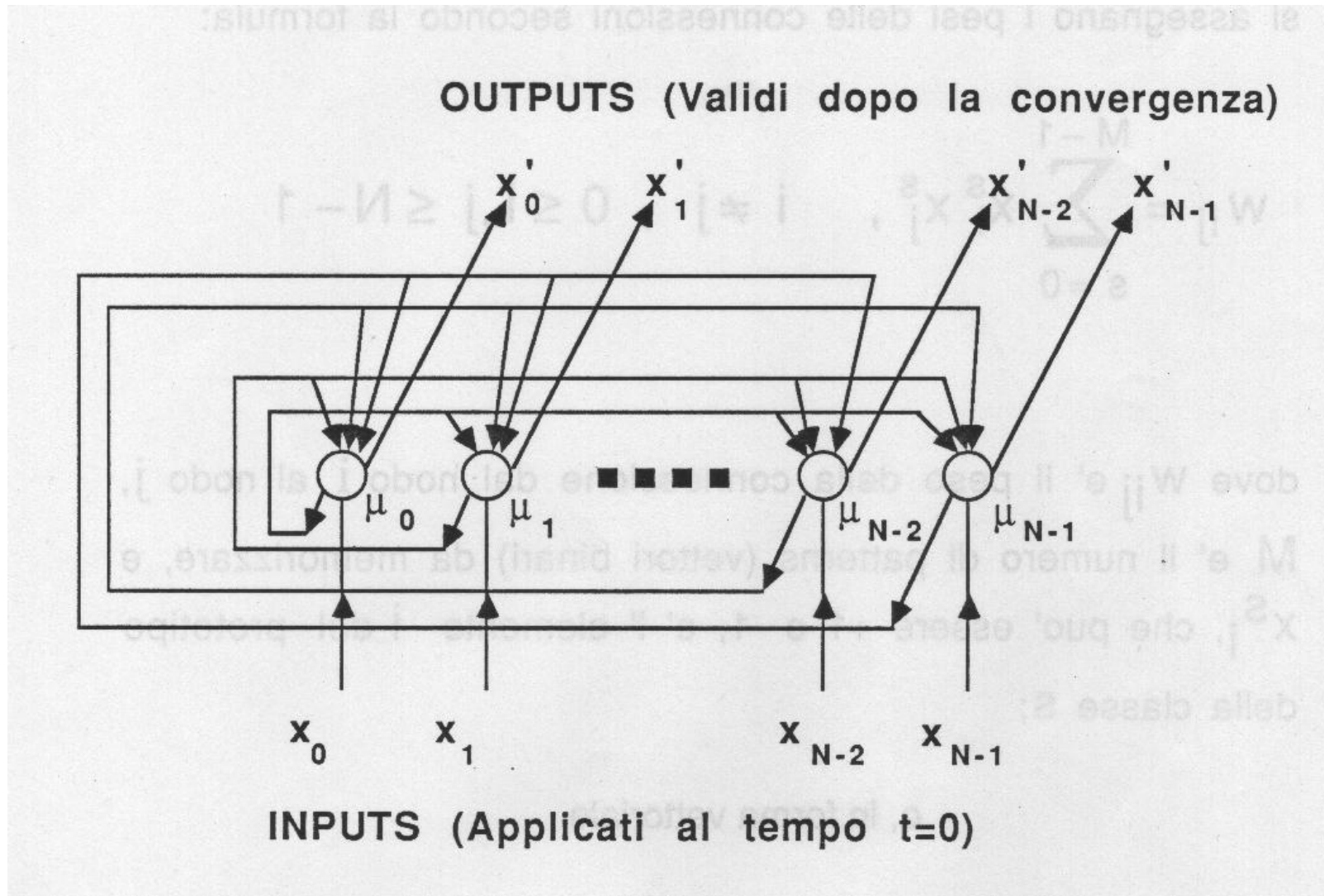


# RETI NEURALI (II PARTE)

# HOPFIELD Neural Net



- è utilizzata come MEMORIA ASSOCIATIVA e come CLASSIFICATORE
- input e output sono BINARI  $\{+1, -1\}$
- i pesi sono fissati con un apprendimento non iterativo (*fixed point learning*)
- al tempo  $t=0$  si dà in input un pattern binario sconosciuto; la rete itera fino a convergere su dei valori di output che rappresentano il pattern più simile all'input presente nella memoria della rete.

# Algoritmo per le Reti di Hopfield

## 1. Apprendimento (fixed point)

si assegnano i pesi delle connessioni secondo la formula:

$$w_{ij} = \sum_{s=0}^{M-1} x_i^s x_j^s, i \neq j, 0 \leq i, j \leq N-1$$

dove  $w_{ij}$  è il peso della connessione dal nodo  $i$  al nodo  $j$ ,  $M$  è il numero di patterns (vettori binari) da memorizzare, e  $x_i^s$ , che può essere  $+1$  o  $-1$ , è l'elemento  $i$  del prototipo della classe  $S$ .

In forma vettoriale:

$$\mathbf{W} = \sum_{s=0}^{M-1} \mathbf{X}_s \mathbf{X}_s^T - \mathbf{1}$$

esempio:

per memorizzare il vettore  $[+1 \ -1 \ +1 \ +1]$  accorrerà una rete con 4 unità, la cui matrice dei pesi è calcolata nel modo seguente:

$$\begin{bmatrix} +1 \\ -1 \\ +1 \\ +1 \end{bmatrix} \begin{bmatrix} +1 & -1 & +1 & +1 \end{bmatrix} -$$

$$- \begin{bmatrix} +1 & 0 & 0 & 0 \\ 0 & +1 & 0 & 0 \\ 0 & 0 & +1 & 0 \\ 0 & 0 & 0 & +1 \end{bmatrix} =$$

$$= \begin{bmatrix} 0 & -1 & +1 & +1 \\ -1 & 0 & -1 & -1 \\ +1 & -1 & 0 & +1 \\ +1 & -1 & +1 & 0 \end{bmatrix}$$

## 2. Elaborazione di un pattern

Lo scopo è di riottenere dalla rete il pattern più simile a quello in input, fra quelli memorizzati.

2.1 si inizializzano le unità con il pattern di input

$\mu_i(0) = x_i$  ,  $0 \leq i \leq N-1$  o, in forma vettoriale  $\mu(0) = \mathbf{x}$

dove  $\mu_i(t)$  è lo stato di attivazione (= output in questo modello) dei nodo  $i$  al tempo  $t$  e  $x_i \in \{-1, +1\}$  è l'elemento  $i$ -esimo del vettore di input.

2.2 si itera fino alla convergenza

$$\mu_j(t+1) = f_h \left[ \sum_{i=0}^{N-1} w_{ij} \mu_i(t) \right], 0 \leq j \leq N-1$$

la funzione  $f_h$  è una funzione non lineare a soglia.  
Il processo viene iterato finché l'output dei nodi rimane immutato con ulteriori iterazioni. I nodi di output rappresentano il pattern memorizzato nella rete che meglio si accorda con l'input dato.



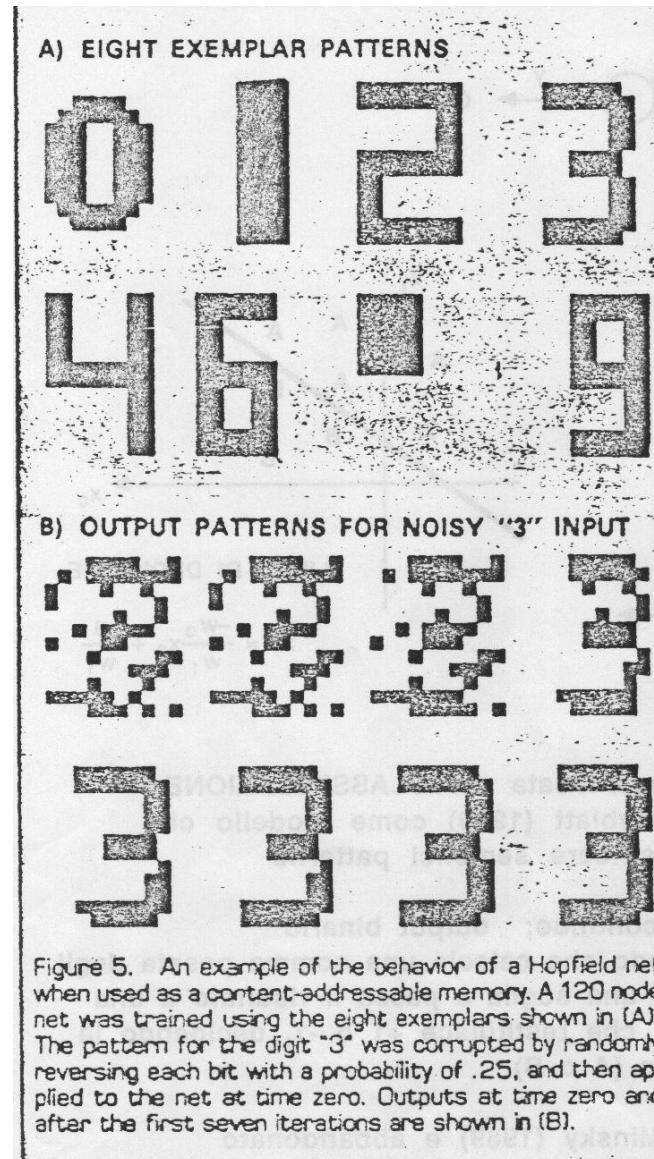
- Quando la rete è usata come Memoria Associativa, l'output (dopo la convergenza) costituisce l'intero pattern cercato.
- Quando è usata come classificatore, bisogna confrontarlo con i prototipi delle  $M$  classi per decidere la classificazione.

# Reti di Hopfield - osservazioni

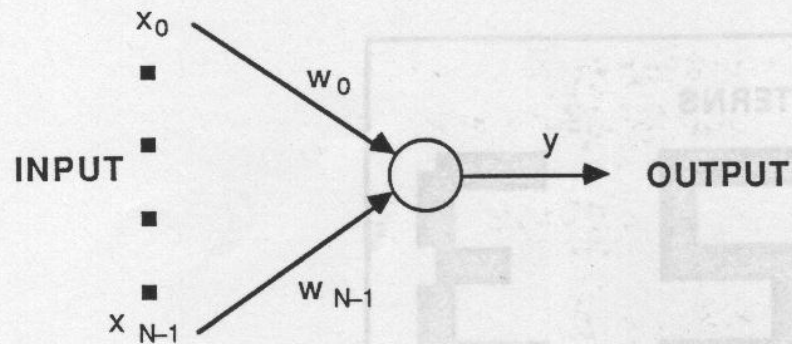
- la rete di Hopfield:
  - apprende col *prodotto esterno* (azzerando la diagonale)
  - accede alla memoria *iterando il prodotto interno* soglia con  $\text{Thr}(x)$
- è più potente di un associatore lineare in quanto è in grado di memorizzare vettori non ortogonali (devono però essere linearmente indipendenti).

- tuttavia il numero di vettori che può essere memorizzato e riottenuto è limitato dal vincolo dell'indipendenza lineare: in media si può memorizzare un numero di patterns pari a  $0.15 N$  ( $N$  = numero di nodi della rete)
- se si memorizzano vettori troppo simili (linearmente dipendenti) il pattern corrispondente diventa instabile, e non è più possibile riottenerlo.

# ESEMPIO DI MEMORIA ASSOCIATIVA CON LA RETE DI HOPFIELD

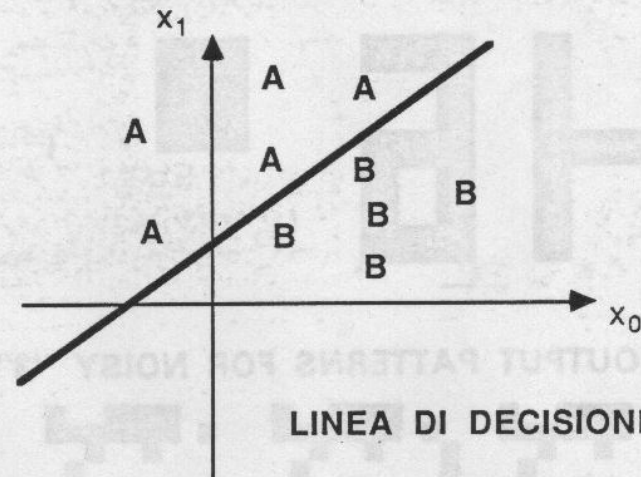
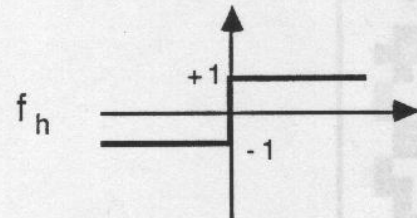


# PERCEPTRON



$$y = f_h \left( \sum_{i=0}^{N-1} w_i x_i - \theta \right)$$

$$y = \begin{cases} +1 \Rightarrow \text{CLASS A} \\ -1 \Rightarrow \text{CLASS B} \end{cases}$$



$$x_1 = \frac{-w_0}{w_1} x_0 + \frac{\theta}{w_1}$$

- è una semplice rete usata per CLASSIFICAZIONE;
- proposto da Rosenblatt (1959) come modello che apprende a classificare semplici patterns;
- input binario o continuo; output binario;
- ha un singolo nodo che calcola una somma pesata degli input, gli sottrae una soglia e passa il risultato a una funzione scalino che restituisce +1 o -1, decidendo la classe dei pattern (A o B);
- fu criticato da Minsky (1969) e abbandonato.

# Algoritmo di Apprendimento del PERCEPTRON

## 1. Inizializzazione di pesi e soglie

Si assegnano a  $w_i(0)$  ( $0 \leq i \leq N-1$ ) e  $\theta$  piccoli valori casuali.  $w_i(t)$  è il peso sulla connessione dall'input  $i$  al tempo  $t$  e  $\theta$  è la soglia del nodo di output.

## 2. Presentazione di un nuovo input e dell'output desiderato:

Si presenta un vettore in input (valori reali)  $\mathbf{x} = x_0, x_1, \dots, x_{N-1}$  insieme con l'output desiderato  $d(t)$  (corretta classificazione)

3. Calcolo dell'output corrente:

$$y(t) = f_h \left( \sum_{i=0}^{N-1} w_i(t) \cdot x_i(t) - \theta \right)$$



4. Modifica dei pesi (*delta rule*):

$$w_i(t+1) = w_i(t) + \eta \cdot [d(t) - y(t)] \cdot x_i(t), 0 \leq i \leq N-1$$

-1 se l'esempio è di classe B

$$d(t) = \{$$

+1 se l'esempio è di classe A

in queste equazioni  $\eta$  è un coefficiente positivo  $< 1$   
e  $d(t)$  è l'output corretto (desiderato) per l'input  
corrente. Si noti che i pesi rimangono immutati se  
la rete prende una decisione corretta.

5. Iterazione ripartendo dal passo 2

# PERCEPTRON - osservazioni

- Rosenblatt ha dimostrato che, se l'input proviene da due classi separabili, l'algoritmo di apprendimento del Perceptron converge e posiziona l'*iperpiano discriminante* fra le due classi.
- spesso, però, due classi non sono separabili da un iperpiano, come il classico caso dell' OR Esclusivo proposto da Minsky come esempio di limite dei Perceptron.

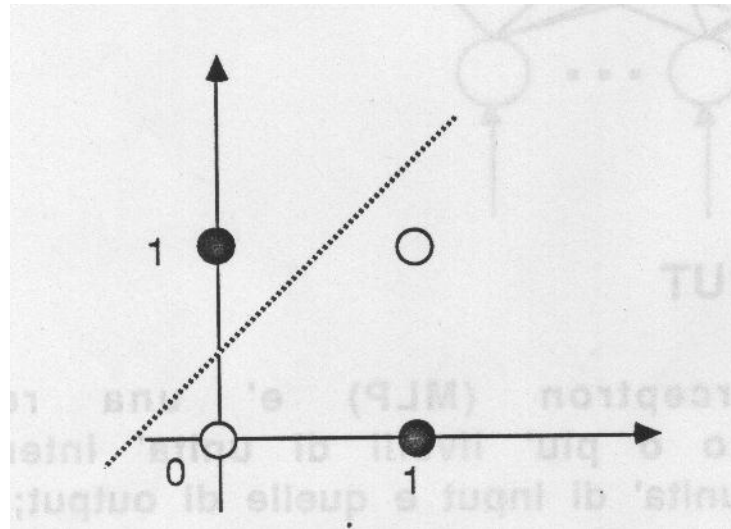
- in tal caso la procedura di apprendimento oscilla, o, se modificata, trova una soluzione che minimizza l'errore quadratico medio fra gli output voluti e quelli ottenuti.
- Il problema dei perceptron è l'eccessiva semplicità del modello. Si capì presto, infatti, che un perceptron a più livelli avrebbe avuto una maggior potenza discriminante. Non vi era però alcun algoritmo di apprendimento per il perceptron a più livelli.
- ulteriori studi hanno portato alla dimostrazione della convergenza di un algoritmo di apprendimento per il perceptron a più livelli (Rumelhart et alii, 1986)

# Il problema dell'OR Esclusivo (XOR)

Apprendere a discriminare le due classi:

classe 0 =  $\{(0, 0), (1, 1)\}$

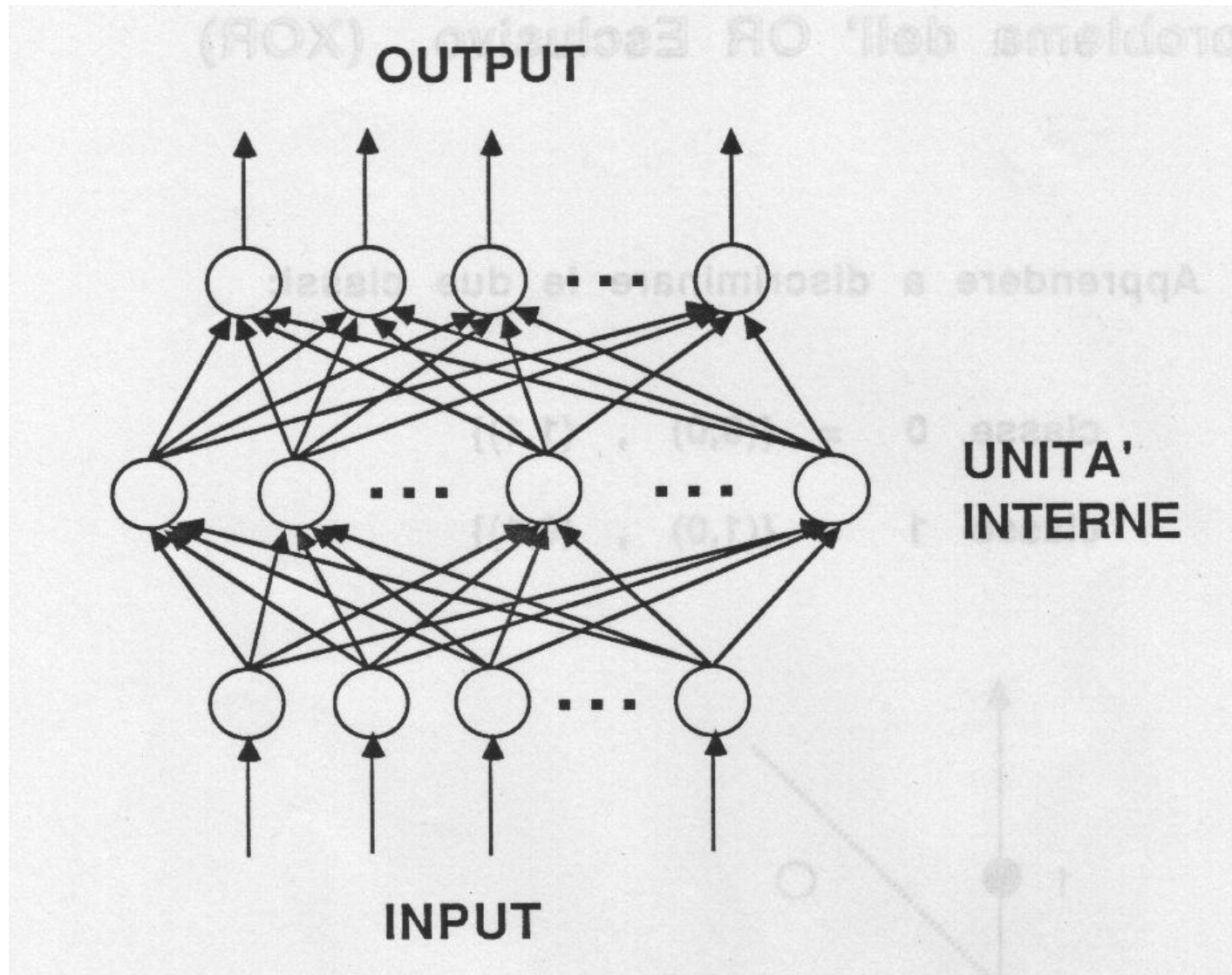
classe 1 =  $\{(1, 0), (0, 1)\}$



non è possibile separare le due classi con una retta.

quindi il PERCEPTRON non è in grado di apprendere a discriminare le due classi dello XOR.

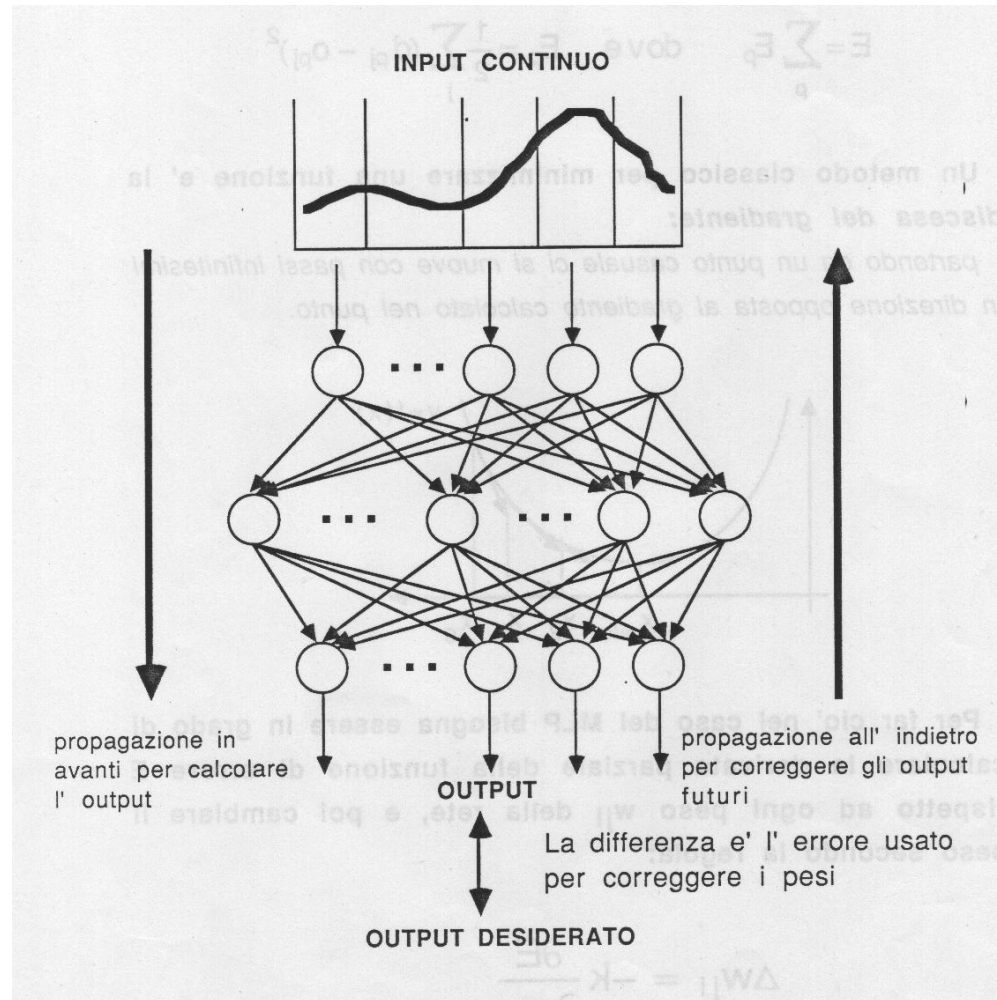
# MULTI-LAYER PERCEPTRON



- Il Multi-Layer Perceptron (MLP) è una rete feed-forward con uno o più livelli di unità interne (hidden units) fra le unità di input e quelle di output; le unità di ogni livello sono connesse con tutte quelle del livello successivo, senza retroazioni;
- input continuo, output continuo  $\in (0, 1)$ , funzione di attivazione Sigmoidale;
- recentemente (1986) è stato proposto un algoritmo di apprendimento (Back Propagation);
- il MLP supera le limitazioni del Perceptron, ed è promettente per varie applicazioni (classificazione, riconoscimento, controllo, sistemi esperti non rule-based);



# MLP: BACKWARD ERROR PROPAGATION

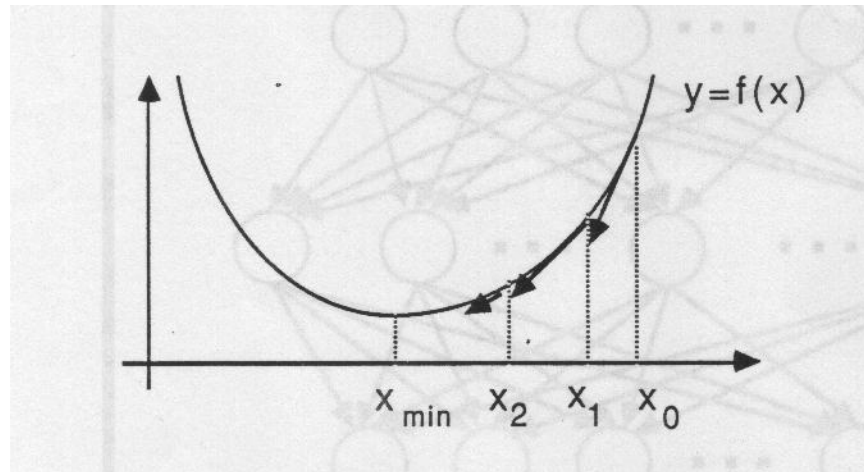


# Back Error Propagation

Con la BP si modificano i pesi del MLP in modo da minimizzare la funzione di errore:

$$E = \sum_p E_p \quad \text{dove:} \quad E_p = \frac{1}{2} \sum_j (d_{pj} - o_{pj})^2$$

Un metodo classico per minimizzare una funzione è la *discesa del gradiente*: partendo da un punto casuale ci si muove con passi infinitesimi in direzione opposta al gradiente calcolato nel punto.



Per far ciò nel caso del MLP bisogna essere in grado di calcolare la derivata parziale della funzione di errore  $E$  rispetto ad ogni peso  $w_{ji}$  della rete, e poi cambiare il peso secondo la regola:

$$\Delta w_{ji} = -k \frac{\partial E}{\partial w_{ji}}$$

calcolando la derivata si ottiene:

$$\Delta_p w_{ji} = \eta \delta_{pi} o_{pj}$$

il peso di una connessione deve quindi essere cambiato in modo proporzionale al prodotto di  $\delta$  (errore sull'unità di arrivo) per  $a_i = o_i$  (valore di attivazione dell'unità di partenza).

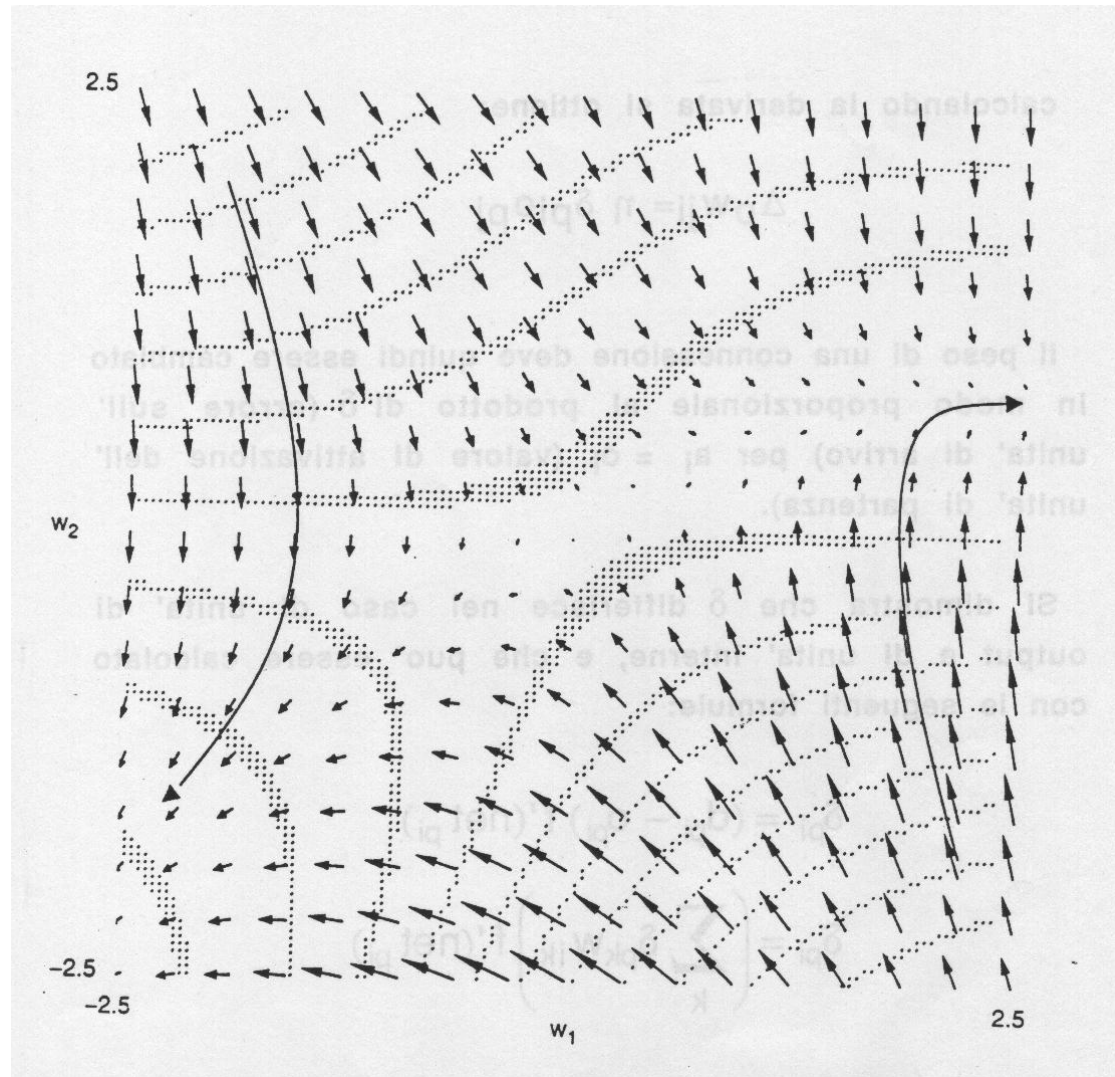
Si dimostra che  $\delta$  differisce nel caso di unità di output e di unità interne, e che può essere calcolato con le seguenti formule:

$$\delta_{pi} = (d_{pi} - o_{pi})f'(\text{net}_{pi})$$

$$\delta_{pi} = \left( \sum_k \delta_{pk} w_{ik} \right) f'(\text{net}_{pi})$$

dove  $\delta_{pi}$  rappresenta la componente dell'errore sull'output imputabile all'unità i-esima (per il pattern p).

# VETTORI OPPOSTI AI GRADIENTI DI UNA SUPERFICIE DI ERRORE BIDIMENSIONALE IN UN MLP CON DUE PESI.



# **Algoritmo di apprendimento del MLP (Back-Propagation)**

*È un algoritmo che minimizza l'errore quadratico medio fra l'output corrente e quello desiderato di un MLP. Procede modificando i pesi in modo da muoversi in direzione opposta al gradiente della funzione di errore (calcolato rispetto ai pesi).*

1. Inizializzazione di pesi e soglie

i pesi e le soglie sono inizializzati a piccoli valori casuali.

2. Presentazione di un esempio (vettore di input e corrispondente vettore di output):

l'input è un vettore di numeri reali  $x_0, x_1, \dots, x_{N-1}$  a cui corrisponde un vettore di output desiderati  $d_0, d_1, \dots, d_{M-1}$ .

Se la rete è usata come classificatore, la classificazione è codificata usando tante unità di output quante sono le classi e ponendo l'output corrispondente alla classe corretta a 1 e tutti gli altri a 0.



### 3. Calcolo degli output correnti:

dato un pattern  $p$ , procedendo a livelli si calcola l'input di ogni nodo con la formula:

$$\text{net}_{pi} = \sum_j w_{ji} o_j + \theta$$

poi si calcola l'output  $o_{pi}$ :

$$o_{pi} = a_{pi} = f(\text{net}_{pi})$$

con

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

#### 4. Modifica dei pesi:

Si procede a livelli, calcolando l'errore sul livello di output e propagandolo all'indietro fino al primo livello interno; quindi si aggiornano i pesi con la *delta rule generalizzata*:

$$\Delta_p w_{ji} = \eta \delta_{pi} o_{pj}$$

che modifica il peso di una connessione in modo proporzionale all'errore compiuto dall'unità di arrivo ( $\delta_{pi}$ ) e alla grandezza dell'output dell'unità di partenza ( $o_{pj}$ ). L'errore sulle unità viene calcolato con le formule:

$$\delta_{pi} = (d_{pi} - o_{pi})f'(\text{net}_{pi})$$

$$\delta_{pi} = \left( \sum_k \delta_{pk} w_{ik} \right) f'(\text{net}_{pi})$$

$$\text{con } f'(x) = f(x) (1 - f(x))$$

la prima calcola l'errore per il livello di output (in base alla differenza fra l'output corrente e quello desiderato),

e la seconda lo propaga alle unità interne (in base all'errore delle unità del livello superiore e all'intensità delle connessioni).


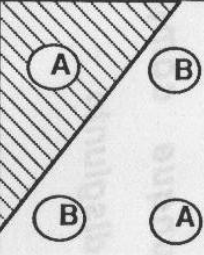
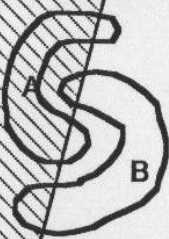
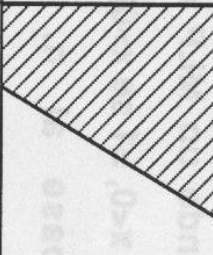
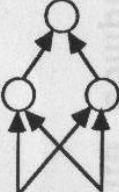
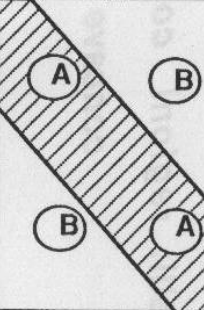
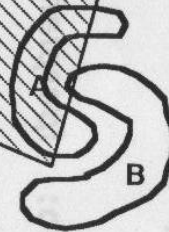
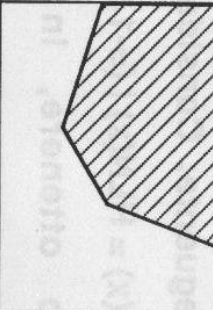
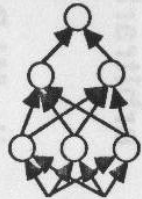
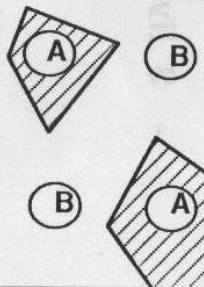
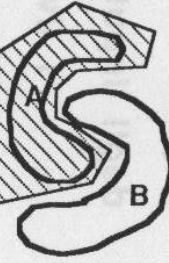
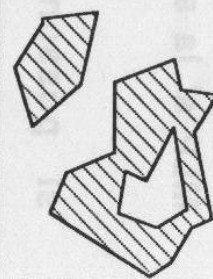
5. Iterazione a partire dal passo 2.

Nel seguito verranno approfonditi i calcoli per la BP.

# MLP - osservazioni

- una interessante osservazione (Lippmann, 1987) sul MLP è la seguente: Considerando un MLP con  $f(x) = |\text{HLN}(X)|$  ( $=-1$  se  $x < 0$ ,  $=1$  se  $x > 0$ ) si possono ottenere, in base al numero di livelli, le seguenti superfici di discriminazione nello spazio degli input:
  - 1 livello: *iperpiano*
  - 2 livelli: *regione convessa*
  - 3 livelli: *regioni comunque complesse, anche concave e disgiunte*
- un comportamento analogo, ma più complesso, si ha usando la sigmoide.

# REGIONI DI DECISIONE FORMATE DA DIVERSI PERCEPTRON

STRUTTURA	TIPO DI REGIONE DI DECISIONE	OR ESCLUSIVO	CLASSI CON REGIONI INFRAMMEZZATE	REGIONI COMUNQUE COMPLESSE
<b>1 LIVELLO</b> 	SEMISPAZIO LIMITATO DA UN IPERPIANO			
<b>2 LIVELLI</b> 	REGIONI CONVESSE APERTE O CHIUSE			
<b>3 LIVELLI</b> 	REGIONI ARBITRARIE (complessita' limitata dal numero dei nodi)			

- quindi, il numero di livelli necessari per formare superfici arbitrariamente complesse è tre.
- il MLP sta ottenendo crescente successo in varie applicazioni (es. pronunzia dell'inglese)
- problemi dei MLP sono:
  - la convergenza è garantita solo a minimi locali;
  - può essere molto lenta, particolarmente con tre livelli.

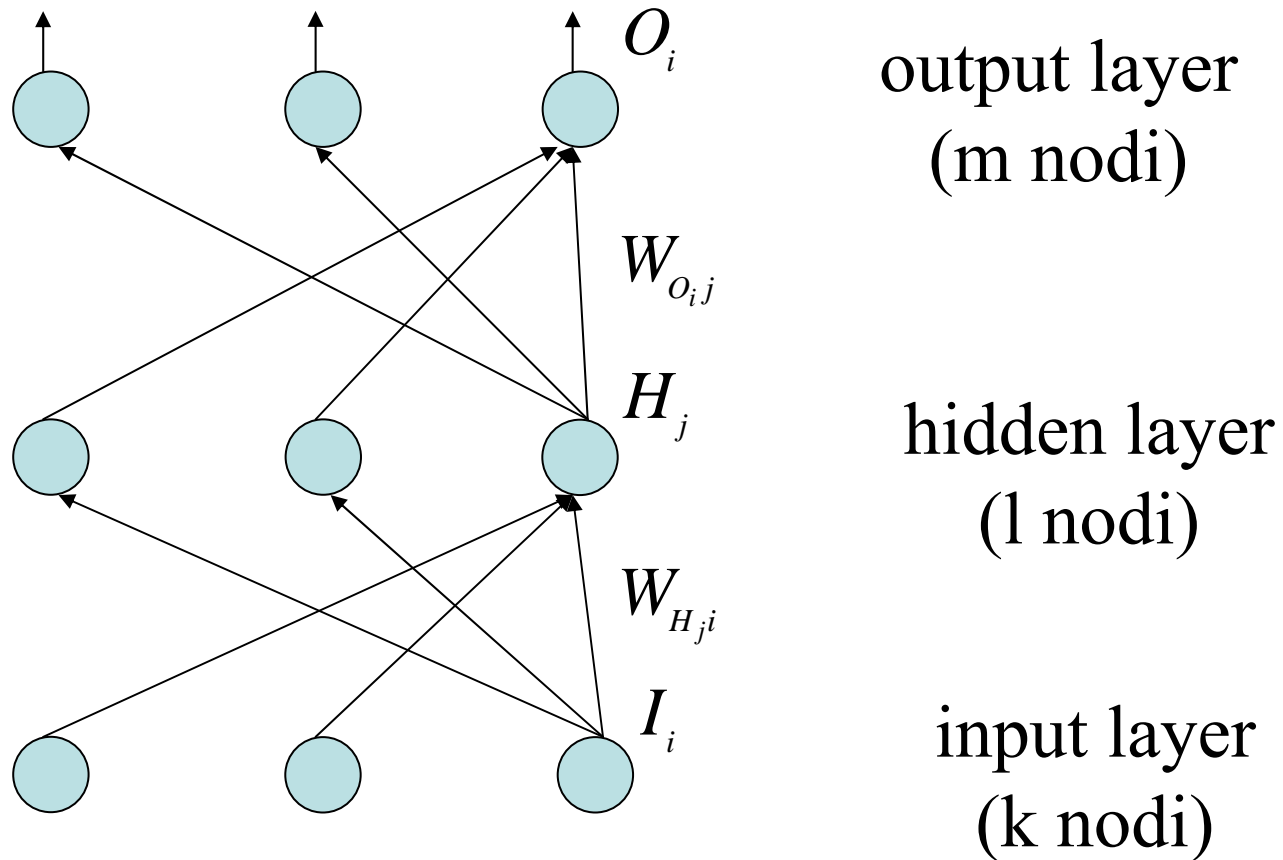


# Come scegliere l'architettura ottima

- Le reti feed-forward sono approssimatori universali, ma una data architettura di rete ha potenzialità limitate.
- La scelta della corretta architettura di rete adatta a risolvere un dato compito è un problema di ricerca nello spazio delle possibili architetture (il problema dell'apprendimento è invece una ricerca nello spazio dei pesi):
  - una rete troppo piccola può essere incapace di risolvere un dato problema;
  - una rete con troppi parametri può creare fenomeni di **over-fitting**, con iper-apprendimento del training set ed incapacità di generalizzazione.

- Possibili soluzioni:
  - calcolo della **VC-dim** dell'architettura per stabilire quanti esempi di un dato dominio sono necessari (in rapporto al numero dei pesi) per ottenere buone capacità di apprendimento e generalizzazione;
  - utilizzo della **cross-validation** e di tecniche *trial-end-error*;
  - utilizzo di **algoritmi genetici** di selezione nello spazio delle architetture;
  - **algoritmi di accrescimento** o *growing*
  - es. algoritmi *Tiling*, *Tower*, *Pyramid* e *Cascade Cor-relation*;
  - **algoritmi di potatura** o *pruning*
  - es. algoritmi *Optimal Brain Damage* e *Optimal Brain Surgeon*

# RETI NEURALI FEED-FORWARD: la Back-Propagation (approfondimento)



Per i nodi hidden vale:

$$H_i(t+1) = \textit{squash}(bias_{H_i} + \sum_{j=1}^{j=k} W_{H_i j} I_j(t))(1)$$

dove:

$$\textit{squash}(x) = \frac{1}{1 + e^{-x}} = f(x)(2)$$

Per i nodi di output, vale:

$$O_i(t+1) = \textit{squash}(\textit{bias}_{o_i} + \sum_{j=1}^{j=l} W_{o_i j} H_j(t))(3)$$

Dato un set fissato e finito di casi input-output, l'errore totale nella performance della rete con un particolare set di pesi può essere calcolato confrontando i vettori di output ottenuti con quelli desiderati per ciascun caso.

L'errore  $E$  è definito come:

$$E = \frac{1}{2} \sum_c \sum_i (O_{i,c} - D_{i,c})^2 \quad (4)$$

dove  $c$  è l'indice sui casi (coppie input-output),  $i$  è l'indice sulle unità di output,  $D$  e  $O$  rispettivamente l'output desiderato e quello ottenuto.

La procedura di addestramento minimizza  $E$  mediante la discesa del gradiente nello spazio dei pesi.

In pratica, si variano i pesi in base alla relazione:

$$\Delta W_{o_{ij}} = -k \frac{\partial E}{\partial W_{o_{ij}}} (5)$$

Questa derivata è semplicemente la somma delle derivate parziali per ciascuno dei casi input-output.

Per ciascun caso, le derivate parziali dell'errore rispetto a ciascun peso sono calcolate durante un passo “backward” che segue un passo “forward” descritto sopra.

Il passo “backward” parte dalle unità di output e propaga all’indietro la derivata dell’errore per ciascun peso, strato per strato.

Per lo strato di output, tenendo conto della (2), si può scrivere che

$$\frac{\partial E}{\partial W_{o_{ij}}} = \frac{\partial E}{\partial o_i} \cdot \frac{\partial o_i}{\partial x_i} \cdot \frac{\partial x_i}{\partial W_{o_{ij}}} \quad (6)$$



Si calcola dapprima la variazione dell'errore rispetto all'output  $O_i$  per ciascuna unità di output  $i$ .

Differenziando l'equazione (4) per un particolare caso,  $c$ , e sopprimendo l'indice  $c$  si ottiene:

$$\frac{\partial E}{\partial O_i} = (O_i - D_i)(7)$$

Per il secondo contributo, occorre tener presente che:

$$f(x) = \frac{1}{1 + e^{-x}} \quad \text{e}$$

$$\begin{aligned} \frac{df}{dx} &= - \frac{-e^{-x}}{(1 + e^{-x})^2} = \frac{e^{-x} + 1 - 1}{(1 + e^{-x})^2} = \\ &= \frac{1}{1 + e^{-x}} - \frac{1}{(1 + e^{-x})^2} = \frac{1}{1 + e^{-x}} \cdot \left( 1 - \frac{1}{1 + e^{-x}} \right) = \\ &= f \cdot (1 - f) \end{aligned}$$

Quindi:

$$\frac{\partial \mathcal{O}_i}{\partial x_i} = O_i (1 - O_i) (8)$$

Infine, poiché è semplicemente una funzione lineare degli stati delle unità hidden e dei pesi delle connessioni a queste unità, è facile calcolare il terzo contributo:

$$\frac{\partial x_i}{\partial W_{O_i j}} = H_j (9)$$

In conclusione, i pesi delle connessioni tra i nodi di output e quelli hidden vengono variati secondo l'espressione:

$$\Delta W_{o_i j} = -k \frac{\partial E}{\partial W_{o_i j}} = -k \cdot (O_i - D_i) \cdot O_i \cdot (1 - O_i) \cdot H_j \quad (10)$$

Per  $bias_{o_i}$  si utilizza la stessa formula, ponendo  $H=1$ .

Nella letteratura, al posto del fattore  $k$ , si trova solitamente  $\eta$ , velocità di addestramento (*learning rate*).

Solitamente si pone:

$$\delta_{O_i} = (D_i - O_i) \cdot O_i \cdot (1 - O_i) \quad (11)$$

e le formule precedenti diventano:

$$\Delta W_{O_i j} = \eta \cdot \delta_{O_i} \cdot H_j \quad (12)$$

e

$$\Delta bias_{O_i} = \eta \cdot \delta_{O_i} \quad (13)$$

In modo del tutto analogo si calcola la variazione dei pesi per le connessioni tra i nodi hidden e i nodi di input (e, in generale, per eventuali strati intermedi):

poiché la topologia è la stessa, la formula sarà del tutto analoga alla (10), sostituendo all'errore in uscita l'errore che si ha nel nodo hidden e tenendo conto che si differenzia rispetto ad  $W_H$  invece che rispetto a  $W_O$ .

Concretamente, l'errore nel nodo hidden è la somma di tutti gli errori provenienti dalle connessioni che escono dal nodo hidden verso i nodi di output. Quindi per un nodo hidden si può porre:

$$E = \frac{1}{2} \sum_{k \in C} e_k^2 \quad (14)$$

dove il neurone  $k$  è un nodo di output.

La (6) qui diventa:

$$\frac{\partial E}{\partial W_{H_i j}} = \frac{\partial E}{\partial H_i} \cdot \frac{\partial H_i}{\partial x_i} \cdot \frac{\partial x_i}{\partial W_{H_i j}} \quad (15)$$

Per la prima componente della derivata parziale vale:

$$\frac{\partial E}{\partial H_i} = \sum_k e_k \frac{\partial e_k}{\partial H_i} \quad (16)$$



dove:

$$e_k = O_k - D_k \quad (17)$$

e, in analogia ai calcoli fatti in precedenza,

$$\frac{\partial e_k}{\partial H_i} = O_k \cdot (1 - O_k) \cdot W_{O_k i} \quad (18)$$

Per le altre due componenti della derivata parziale il calcolo è simile a quello visto in precedenza (cambiano solo i nomi delle variabili).

Quindi complessivamente si ha:

$$\Delta W_{H_i j} = \eta \cdot H_i \cdot (1 - H_i) \cdot I_j \cdot \sum_{k=1}^{k=m} \delta_{O_k} \cdot W_{O_k i} \quad (19)$$

avendo posto

$$\delta_{O_k} = (D_k - O_k) \cdot O_k \cdot (1 - O_k) \quad (20)$$

Nella pratica, per evitare perturbazioni locali, si “filtra” la variazione dei pesi, così:

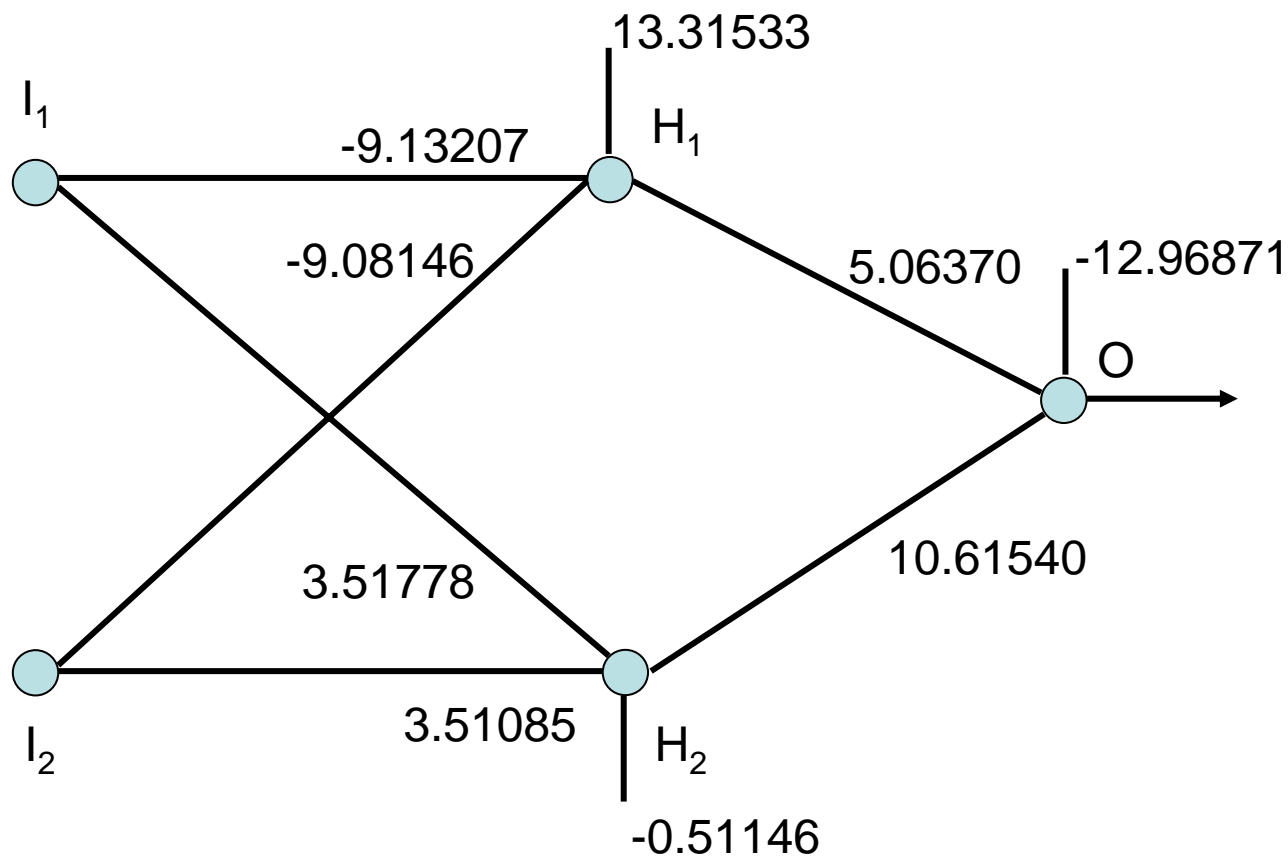
$$\Delta w(t) = -\eta \cdot \frac{\partial E}{\partial w(t)} + \alpha \cdot \Delta w(t-1) \quad (21)$$

dove  $\eta$  è il *learning rate*,  $\alpha$  è detto *momento* (anche se è più propriamente una *viscosità*).

Valori tipici:  $\eta = 0.2$  (o  $0.5$ ),  $\alpha = 0.9$  .

# Considerazioni.

## Rete EX-OR dopo l'addestramento.

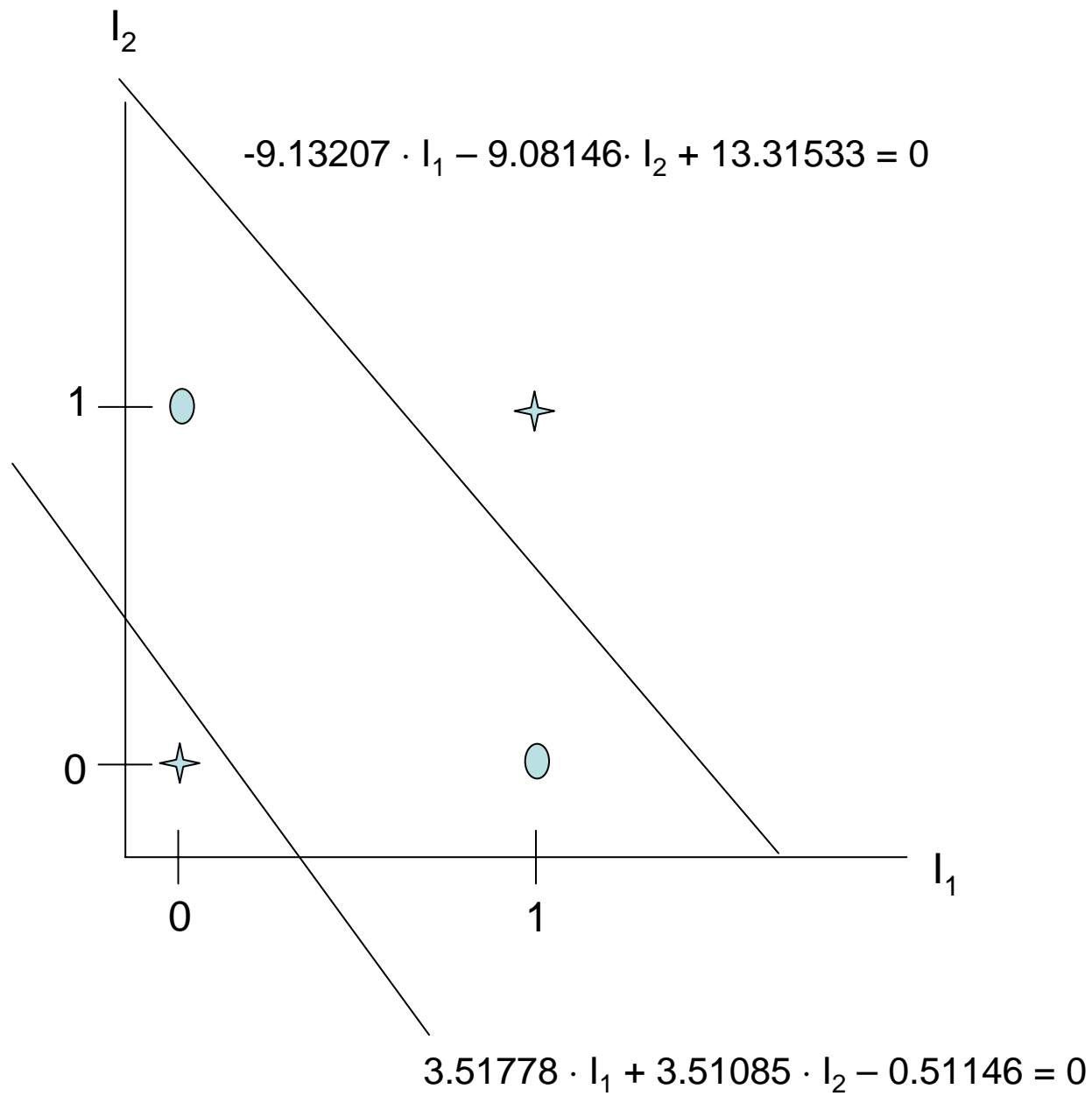


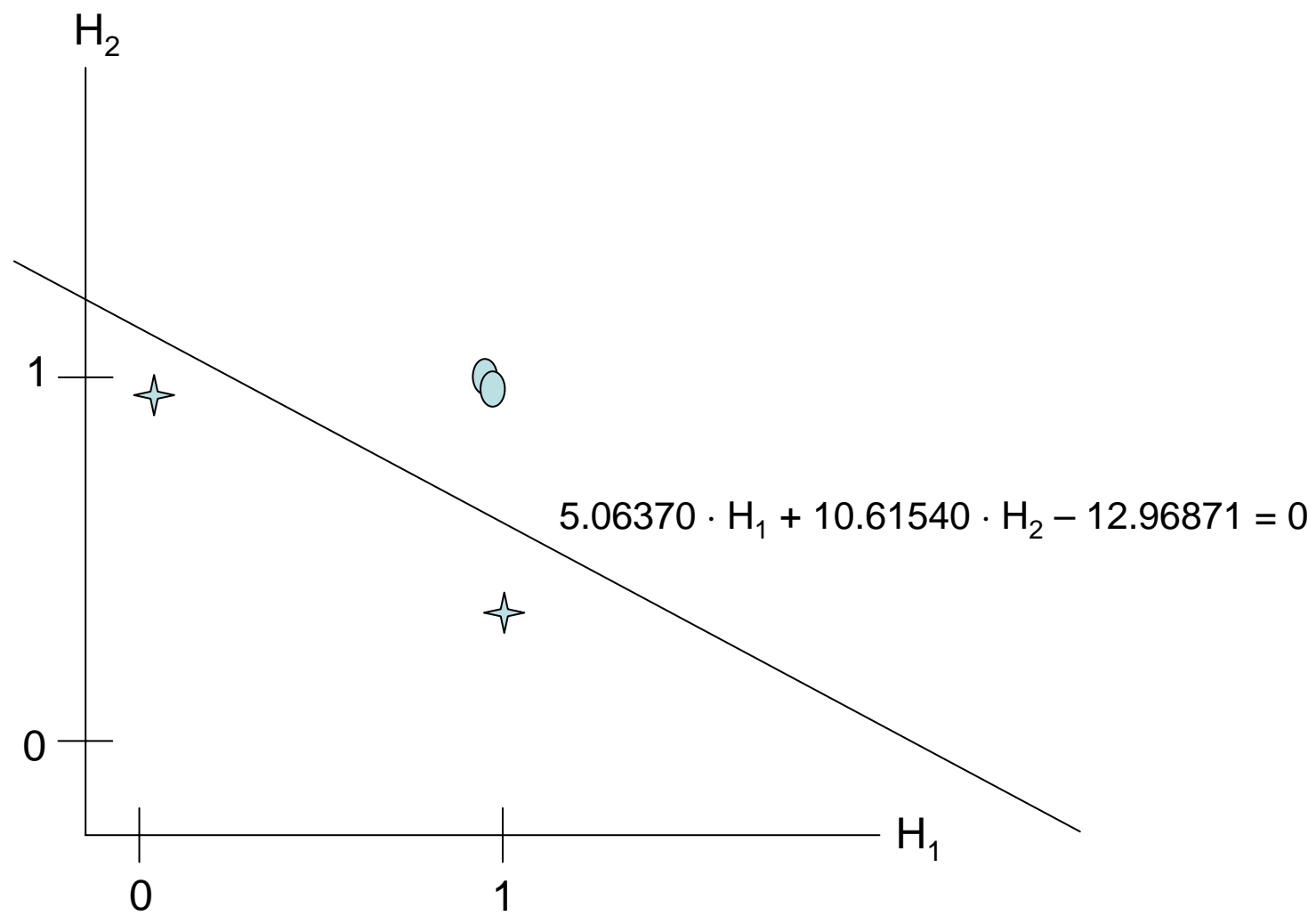
$$H_1 = \text{squash}(-9.13207 \cdot I_1 - 9.08146 \cdot I_2 + 13.31533)$$

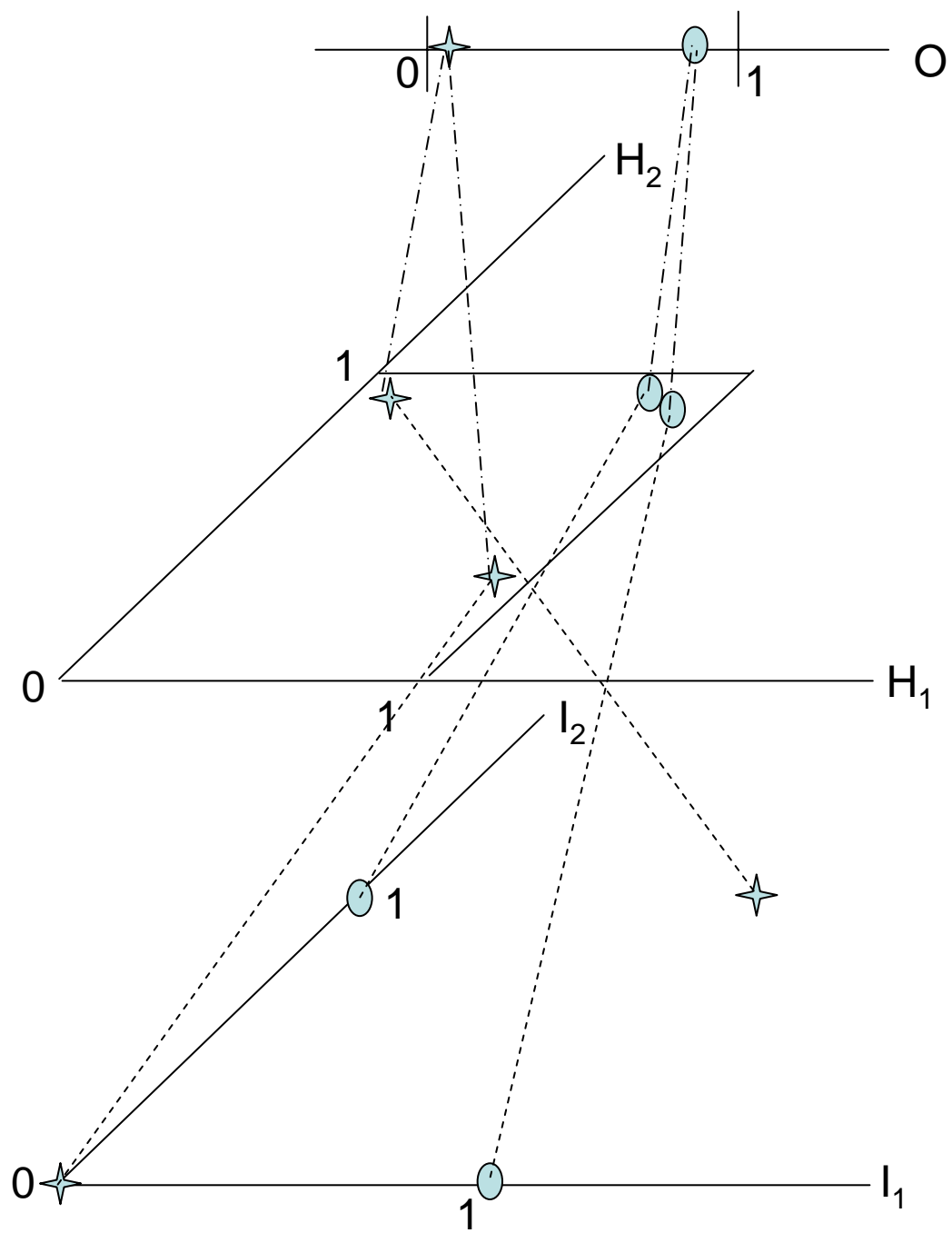
$$H_2 = \text{squash}(3.51778 \cdot I_1 + 3.51085 \cdot I_2 - 0.51146)$$

$$O = \text{squash}(5.06370 \cdot H_1 + 10.61540 \cdot H_2 - 12.96871)$$

I1	I2	H1	H2	O
0	0	1.0000	0.3749	0.0193
0	1	0.9857	0.9525	0.8942
1	0	0.9850	0.9529	0.8942
1	1	0.0074	0.9985	0.0885

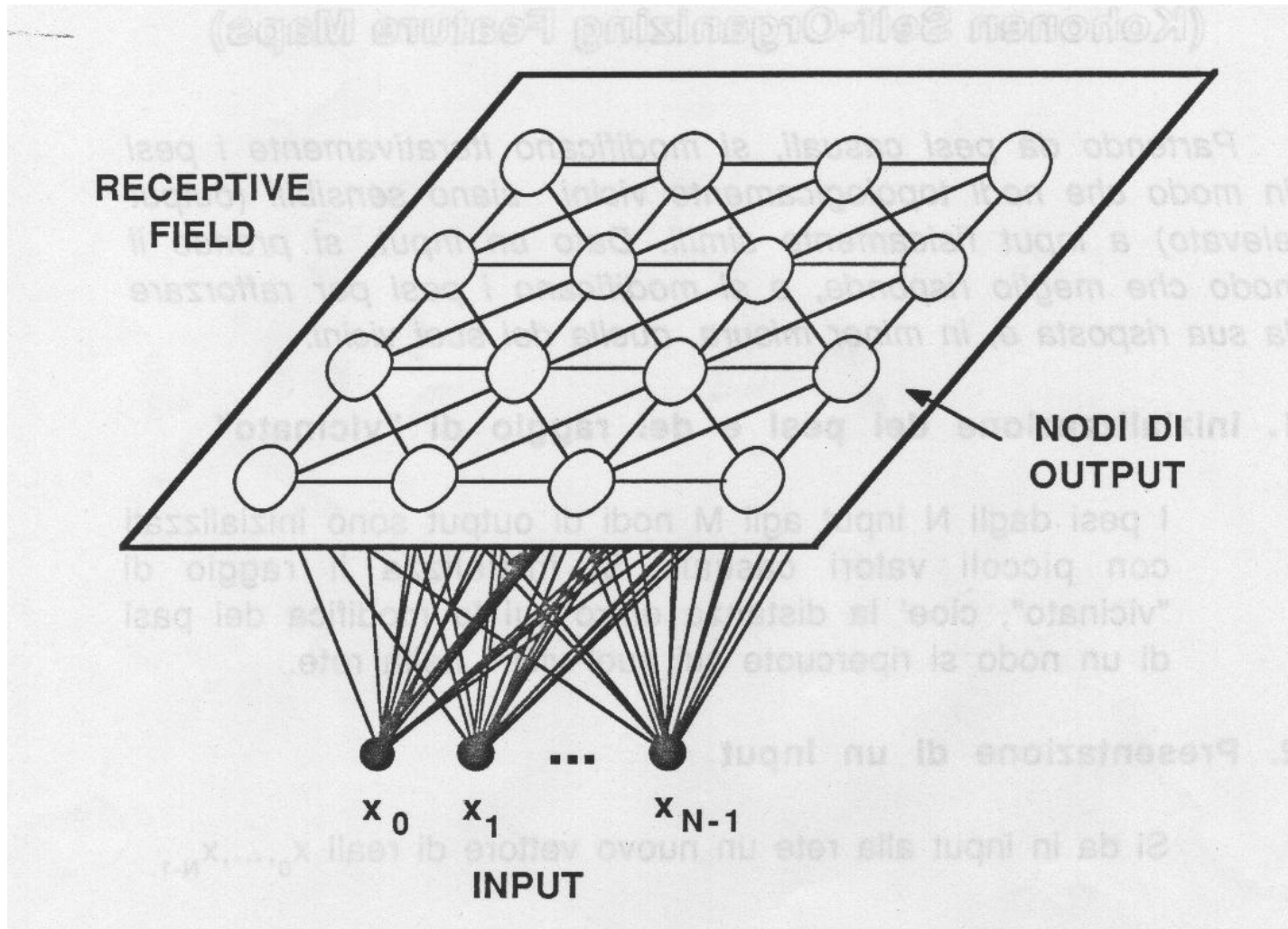






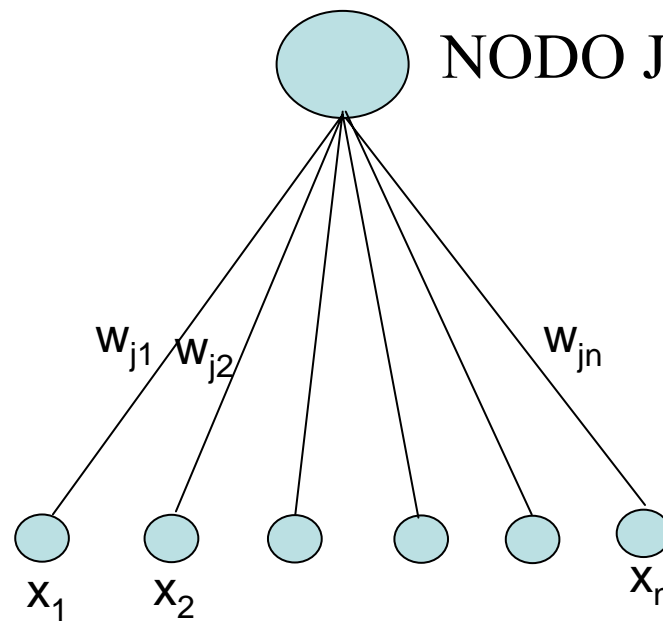


# KOHONEN'S SELF ORGANIZING FEATURE MAPS



Ogni nodo è connesso a tutti gli input.

I pesi  $w_i$  di fatto rappresentano un vettore. È questo vettore che viene confrontato con il vettore di input.



- è una rete neurale che si auto organizza per creare dei raggruppamenti in classi (clusters) di esempi non classificati; è utilizzata per costruire dei Quantizzatori Vettoriali e, in genere, per la compressione di dati;
- la rete (Map) è una griglia bidimensionale di nodi che rappresentano i centri delle classi (campo recettivo);

- i nodi di input sono connessi completamente al campo recettivo; l'input è un vettore di reali (features);
- l'apprendimento è non-supervisionato, cioè gli esempi non sono classificati; il numero di classi è prefissato.

# Algoritmo di Apprendimento per le Reti Auto-Organizzanti

*Partendo da pesi casuali, si modificano iterativamente i pesi in modo che nodi topologicamente vicini siano sensibili (output elevato) a input fisicamente simili. Dato un input, si prende il nodo che meglio risponde, e si modificano i pesi per rafforzare la sua risposta e, in minor misura, quella dei suoi vicini.*

## 1. Inizializzazione dei pesi e del raggio di "vicinato"

I pesi dagli  $N$  input agli  $M$  nodi di output sono inizializzati con piccoli valori casuali.

Si inizializza il raggio di "vicinato", cioè la distanza entro cui la modifica dei pesi di un nodo si ripercuote sui suoi vicini nella rete.

## 2. Presentazione di un input

Si dà in input alla rete un nuovo vettore di reali  $x_0$ ,  
....  $X_{N-1}$ .

## 3. Calcolo della distanza da tutti i nodi

Si calcola la distanza di fra il vettore in input e ogni nodo di output  $j$  con la formula

$$d_j = \sum_{i=0}^{N-1} (x_i(t) - w_{ij}(t))^2$$

dove  $x_i(t)$  è l'input al nodo  $i$  al tempo  $t$  e  $w_{ij}(t)$  è il peso dall'input  $i$  al nodo di output  $j$  al tempo  $t$ .

4. Selezione del nodo di output con la minima distanza  
si seleziona il nodo  $j^*$  con la minima distanza  $d_j$ .

5. Aggiornamento dei pesi verso il nodo e i suoi vicini  
Si modificano i pesi verso  $j^*$  e tutti i nodi del vicinato  
definito da  $NEj^*(t)$  con la formula:

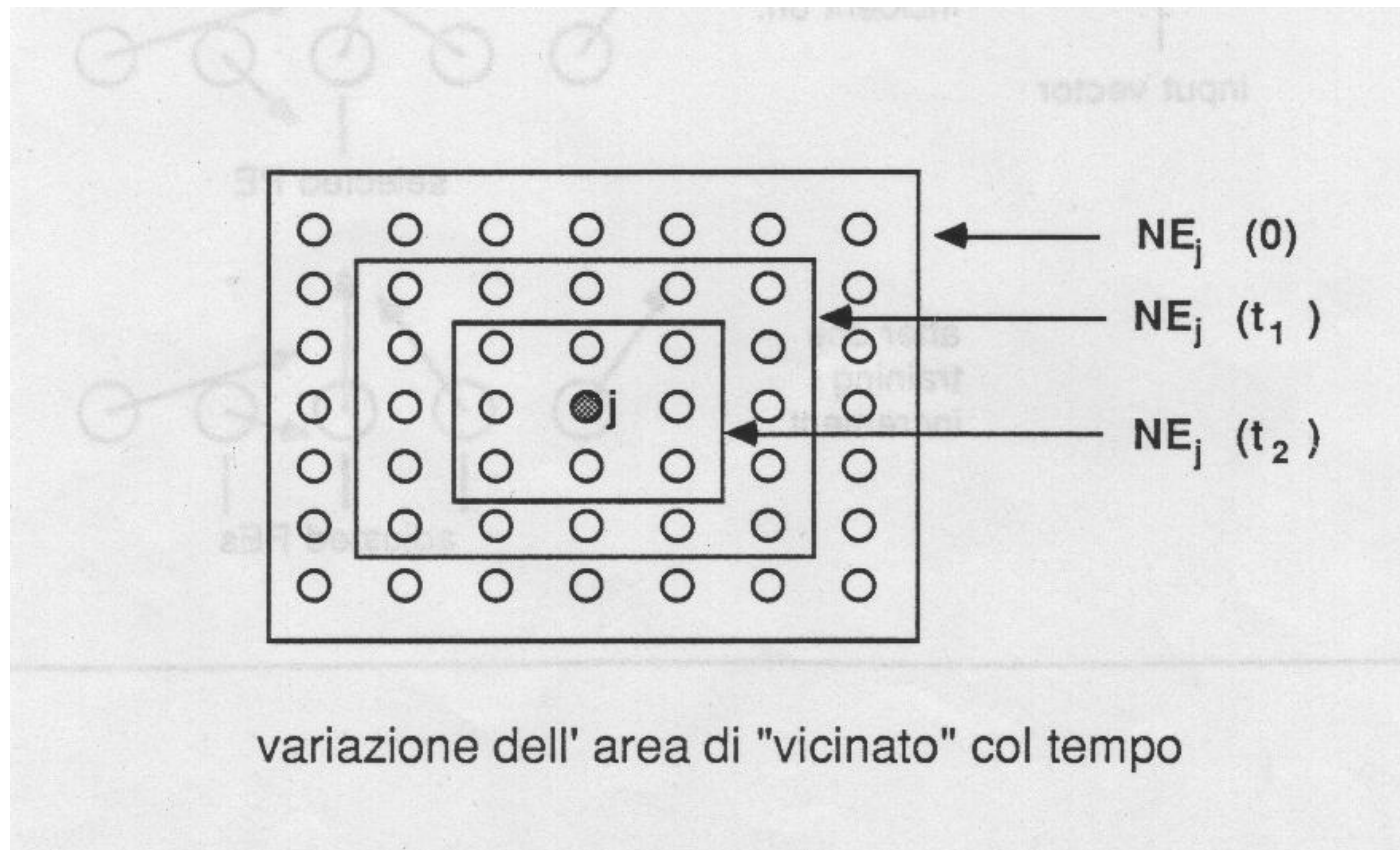
$$w_{ij}(t+1) = w_{ij}(t) + \eta(t) \cdot a(j, j^*) \cdot (x_i(t) - w_{ij}(t))$$

per  $j \in NEj^*(t) \quad 0 \leq i \leq N-1$

dove il termine  $\eta(t)$  è un coefficiente di incremento che  
diminuisce col tempo e  $a(j, j^*)$  è un coefficiente che varia in  
funzione della distanza del nodo  $j$  da  $j^*$ . Anche la  
dimensione del vicinato  $NEj^*(t)$  diminuisce col tempo.



## 6. Iterazione dal passo 2.



# Self-Organization

Da Kohonen, 1984

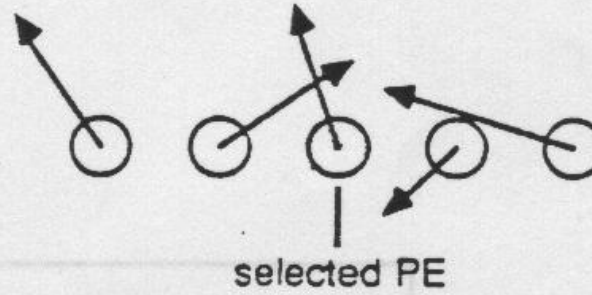
The IWs leading from a vector input channel to a receptive field become ordered in relation to their immediate topological neighbors, as a result of an iterative process that systematically alters IWs in local, selected PE neighborhoods.

- An input vector is incident on a receptive field.
- The PE whose IW vector most closely matches the input vector is selected for modification.

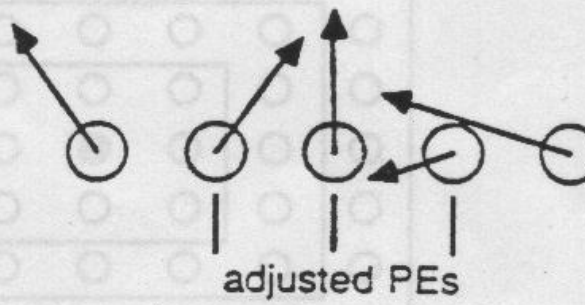
- The IW vector is adjusted to more closely match the input vector.
- The IWs of the neighbors of the selected PE are also adjusted, but to a smaller extent. It is this neighborhood reorientation that makes self-organization work.

input vector

incident on:



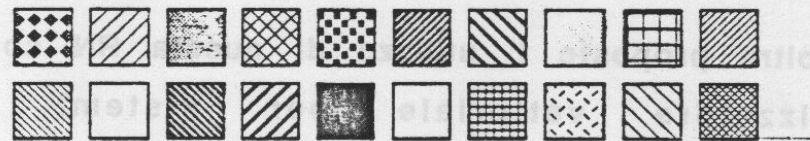
after one  
training  
increment:



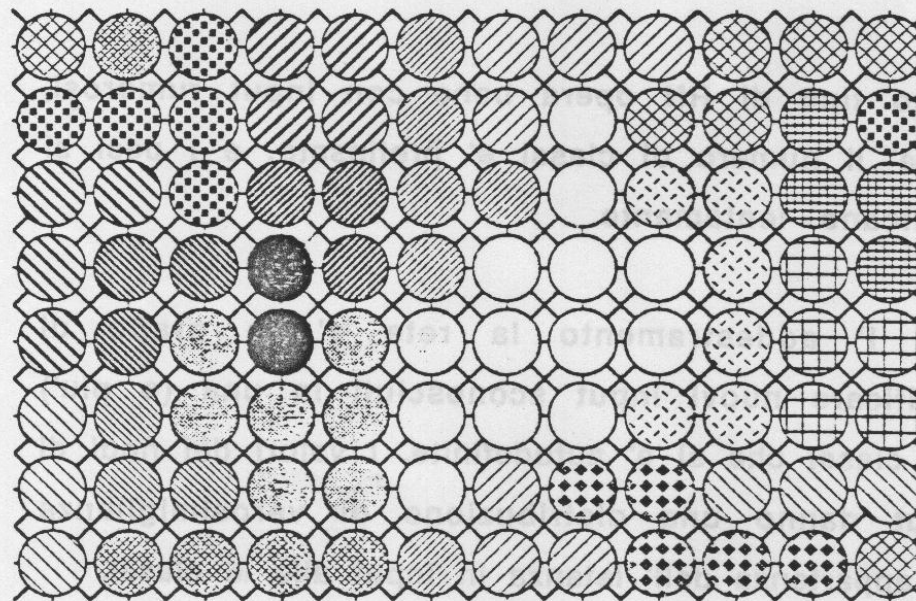
## An Example of Self-Organization

A set of textures can be associated with one another by enumerating their characteristics in a vector representation. Kohonen's self-organizing principle can be applied to this set.

Texture set:



How a self-organizing network may arrange itself:



This field has a toroidal topology, that is, it is a two dimensional array that wraps around at all four edges. The texture in each circle (PE) denotes the closest matching texture vector to that PE's IWs.

# KOHONEN MAPS - osservazioni

- Kohonen (1984) ha dimostrato la convergenza dell'algoritmo di apprendimento e proposto molti esempi di utilizzo.
- Ha inoltre proposto l'utilizzo di questa RN come quantizzatore vettoriale per sistemi di riconoscimento dei parlato e, in genere, per la compressione di dati grezzi in classi significative.
- Questo tipo di RN opera bene con input rumorosi, poiché il numero di classi è prefissato, e i pesi si modificano lentamente.

- Dopo l'addestramento la rete è in grado di classificare nuovi input sconosciuti in una (o più) delle classi che si è autodefinita. I valori dei nodi di output danno una distribuzione di verosimiglianza dell'appartenenza dell'istanza in input alle  $M$  classi.
- tale modello può anche servire, usato in modo opportuno, per problemi di controllo (es. movimento di robot) e di classificazione.