

Reti Neurali

- Fino ad ora abbiamo visto approcci di ragionamento simbolici basati su simboli e regole sintattiche per la loro manipolazione.
- I connessionisti credono che la manipolazione simbolica sia un meccanismo molto povero.
- Gli approcci connessionisti si basano sulla simulazione dei meccanismi presenti nel cervello umano
- Sono pertanto stati sviluppati modelli che assomiglino alle strutture neurologiche.

Il dilemma dell' I.A.

I computer sono eccellenti nel calcolo, ma falliscono quando si cerca di riprodurre attività tipicamente umane:

- Percezione sensoriale
- Coordinamento senso-motorio
- Riconoscimento di immagini
- Capacità di adattamento

Bambino batte Computer

3 a 0

Sebbene un computer possa battere il campione del mondo di scacchi, esso non è in grado di competere con un bambino di 3 anni nel

- costruire con il Lego
- riconoscere il volto di una persona
- riconoscere la voce dei genitori

Problema

- Le azioni complesse dipendono da molti fattori, che non possono essere previsti esattamente in un programma.
- Tali fattori devono essere acquisiti con l'esperienza, in una fase di apprendimento.

La mente ha bisogno di un corpo!

Esempi

- Afferraggio di un oggetto è determinato da numerosi fattori:
 - la posizione dell'oggetto
 - la nostra postura
 - la dimensione e la forma dell'oggetto
 - il peso previsto
 - gli eventuali ostacoli interposti

Riconoscimento del parlato

Richiede una fase di apprendimento necessaria per:

- adattarsi al soggetto che parla
- filtrare i rumori esterni
- separare eventuali altre voci

Riconoscimento di immagini



PERA E MELA

BARBA E BAEEL

Come funziona il cervello?

- Quando riconosciamo un volto o afferriamo un oggetto non risolviamo equazioni.
- Il cervello lavora in modo **associativo**:

ogni stato sensoriale evoca uno stato cerebrale (un'attività elettro-chimica) che viene memorizzata a seconda delle necessità.

Colpire una palla da tennis

- La traiettoria dipende da diversi fattori:
 - forza di lancio, angolazione iniziale, effetto, velocità del vento;
- La previsione della traiettoria richiede:
 - la misurazione precisa delle variabili;
 - la soluzione simultanea di equazioni complesse, da ricalcolare ad ogni acquisizione dei dati.

Come fa un giocatore a fare tutto ciò?

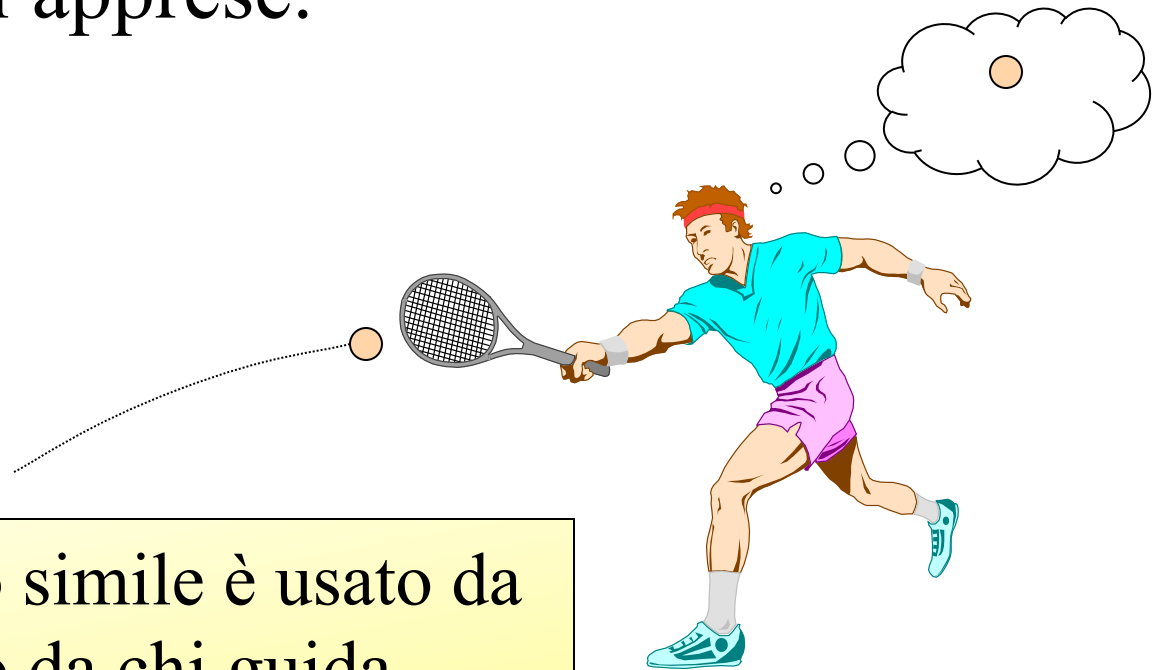
Fase di apprendimento

- In una fase di apprendimento si provano le azioni e si memorizzano quelle buone:
 - se la palla è passata in questa zona del campo visivo, fai un passo indietro;
 - se la palla ...



Fase operativa

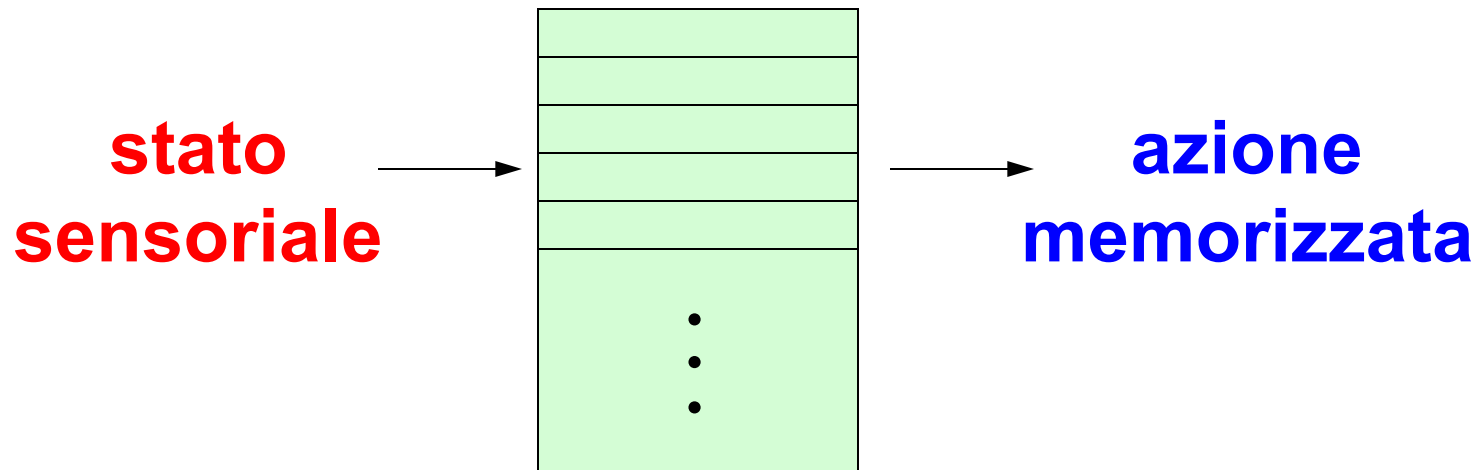
- Una volta allenati, il cervello esegue le azioni *senza pensare*, sulla base delle associazioni apprese.



Un meccanismo simile è usato da
chi suona o da chi guida

Il calcolo associativo

- Un insieme di equazioni complesse vengono risolte mediante una **look-up table**.
- Essa è costruita in base all'esperienza e viene affinata con l'allenamento.



Riconoscimento del parlato

- Avviene in presenza di rumori o di forti distorsioni
- E' indipendente dal soggetto che parla
- E' indipendente dall'accento

Il calcolo neuronale

L'estrema difficoltà di trattare questi problemi con il calcolatore ha fatto nascere l'esigenza di studiare nuove metodologie di calcolo, ispirate alle reti neurali.

Medici → studi sul cervello

Ingegneri → risoluzioni di problemi

Evoluzione della ricerca

- **1943, McCulloch e Pitts:** nasce il primo modello neurale: il neurone binario a soglia.
- **1949, Hebb:** dagli studi sul cervello, emerge che l'apprendimento non è una proprietà dei neuroni, ma è dovuto a una modifica delle sinapsi.
- **1962, Rosenblatt:** propone un nuovo modello di neurone capace di apprendere mediante esempi: il perceptron.

- **1969, Minsky e Papert:** dimostrano i limiti del perceptron: crolla l'entusiasmo sulle reti neurali.
- **1982, Hopfield:** propone un modello di rete per realizzare memorie associative.
- **1982, Kohonen:** propone un tipo di rete auto-organizzante (mappe recettive).
- **1985, Rumelhart, Hinton e Williams:** formalizzano l'apprendimento di reti neurali con supervisione (Back-Propagation).

Alcune proprietà del cervello

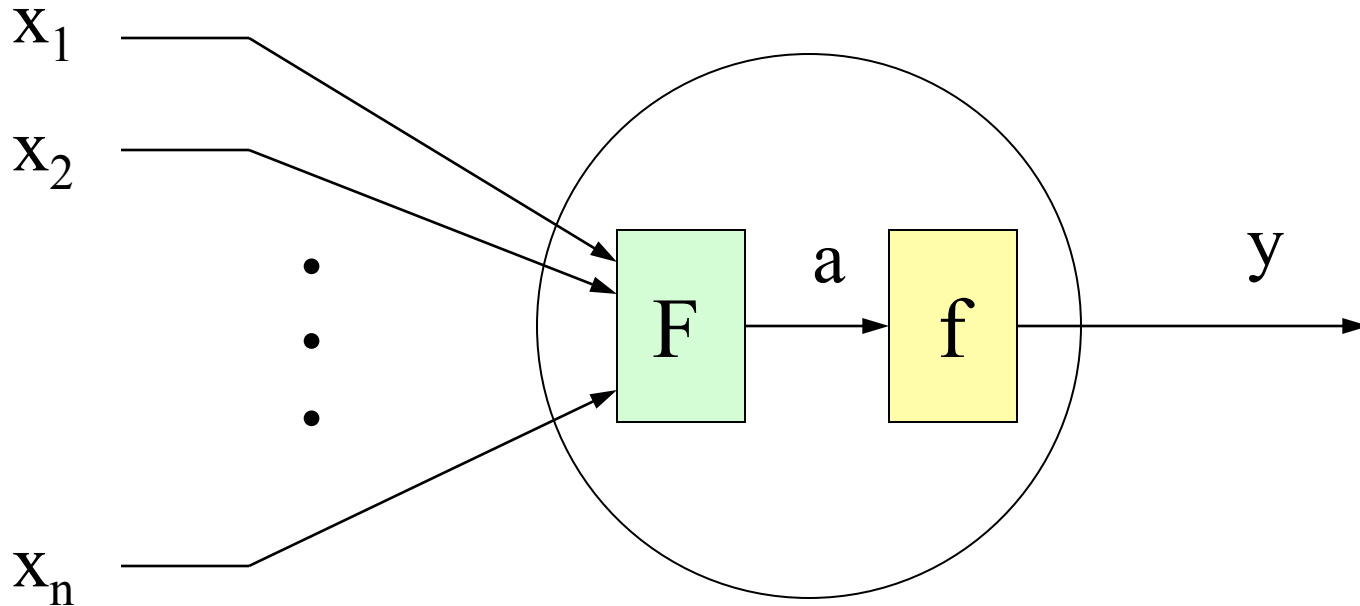
- **Velocità dei neuroni:** alcuni ms
- **Numero di neuroni:** $10^{11} \div 10^{12}$
- **Connessioni:** $10^3 \div 10^4$ per neurone
- **Funzionamento:** attivazione/inibizione
- **Controllo distribuito:** manca una CPU
- **Tolleranza ai guasti:** graceful degradation

Modello di un neurone

Occorre definire

- il numero dei canali d'ingresso: N
- il tipo dei segnali d'ingresso: \mathbf{x}_i
- i pesi delle connessioni: \mathbf{w}_i
- la funzione di attivazione: F
- la funzione di uscita: f

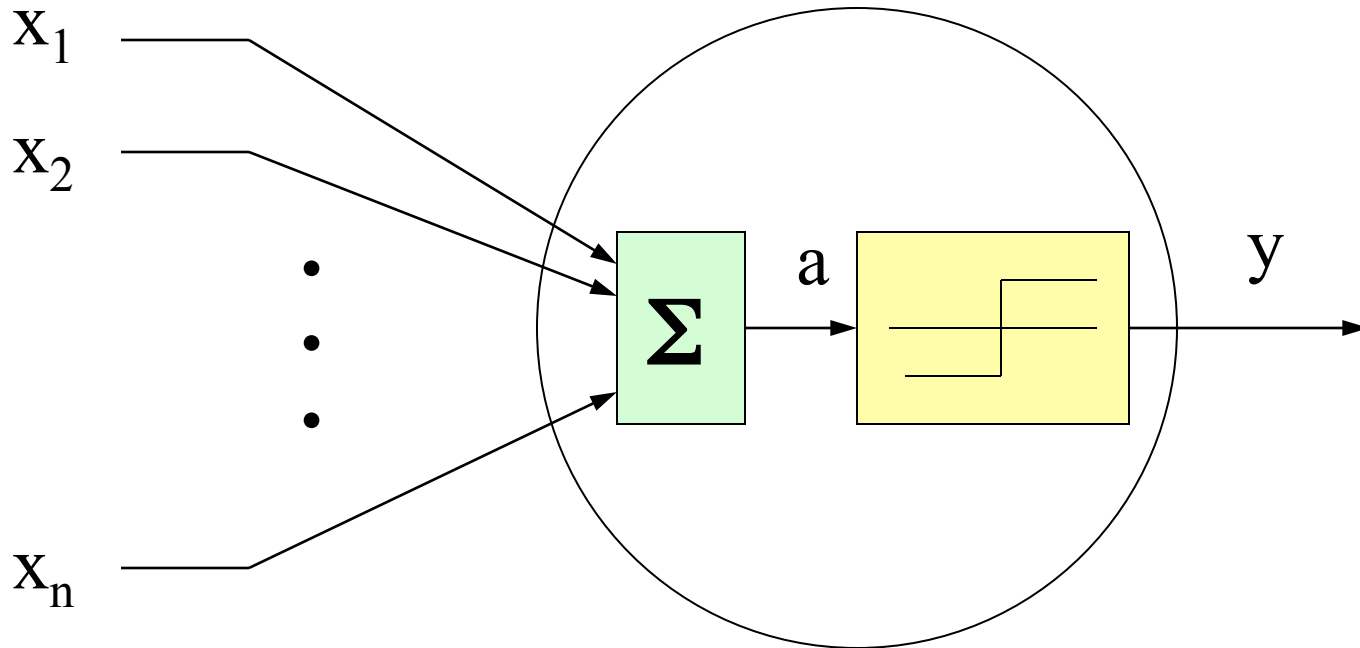
Modello generale di neurone



$$a(t) = F(x_1, x_2, \dots, x_n)$$

$$y(t) = f(a)$$

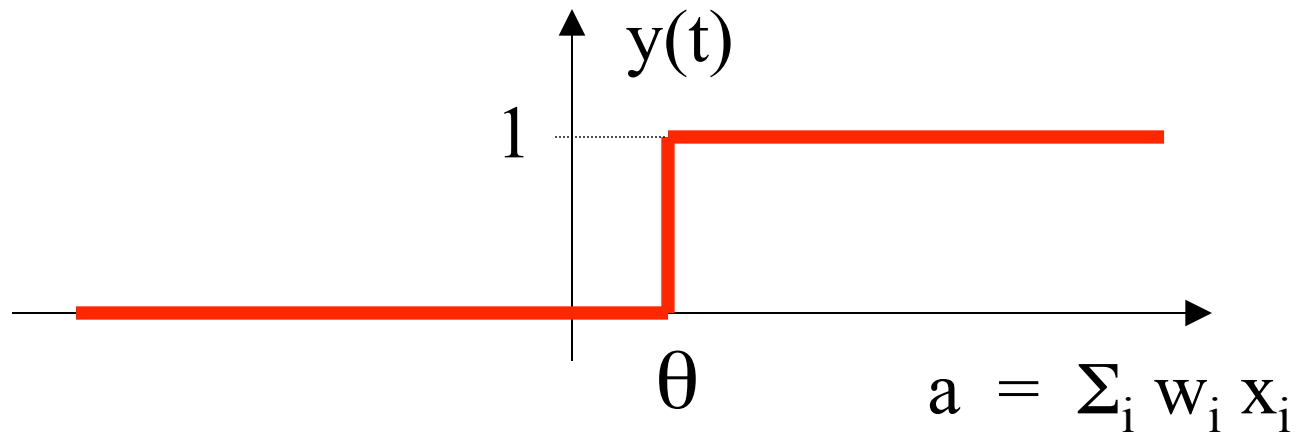
Il neurone binario a soglia



$$a = \sum_i w_i x_i$$
$$y = \text{HS}(a - \theta)$$

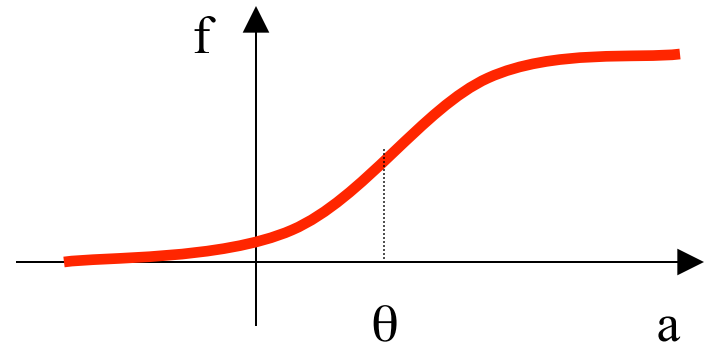
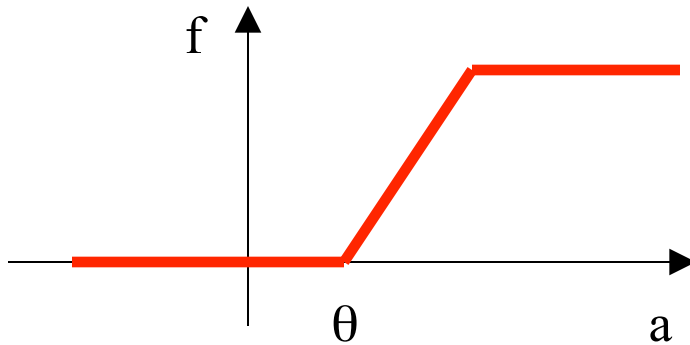
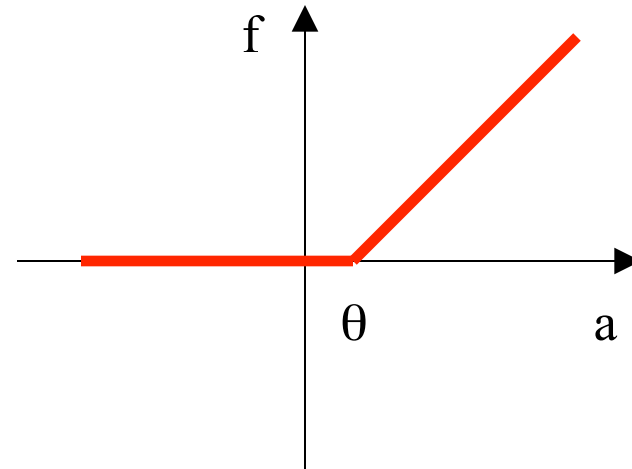
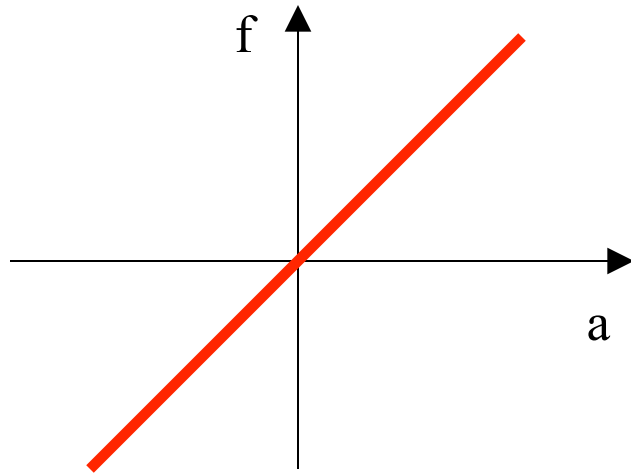
Funzionamento neuronale:
gli impulsi ricevuti dai dendriti
aumentano il potenziale elettrico nel
neurone fino a una certa soglia

Funzione di Heaviside



$$y(t) = \begin{cases} 0 & \text{se } \sum_i w_i x_i < \theta \\ 1 & \text{altrimenti} \end{cases}$$

Altre funzioni di uscita

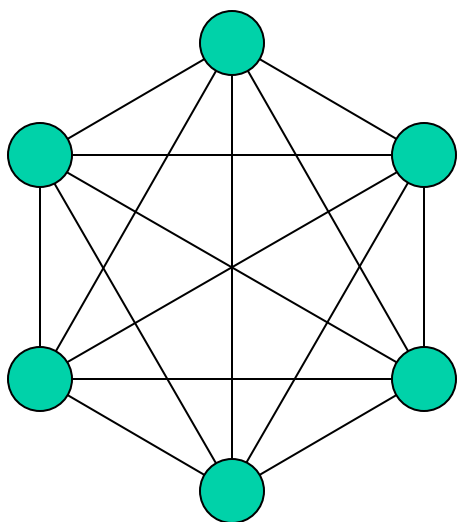


Reti di neuroni

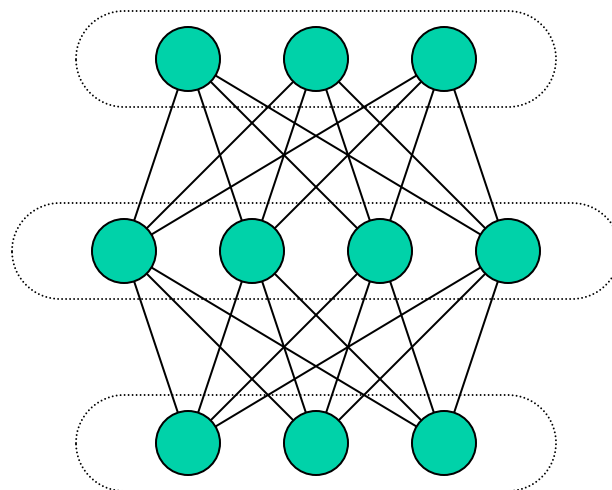
Per costruire una rete neurale occorre definire:

- Il modello dei neuroni
- L'architettura della rete
- La modalità di attivazione dei neuroni
- Il paradigma di apprendimento
- La legge di apprendimento

Architetture di rete



Completamente connessa



stratificata

Rappresentazione delle connessioni

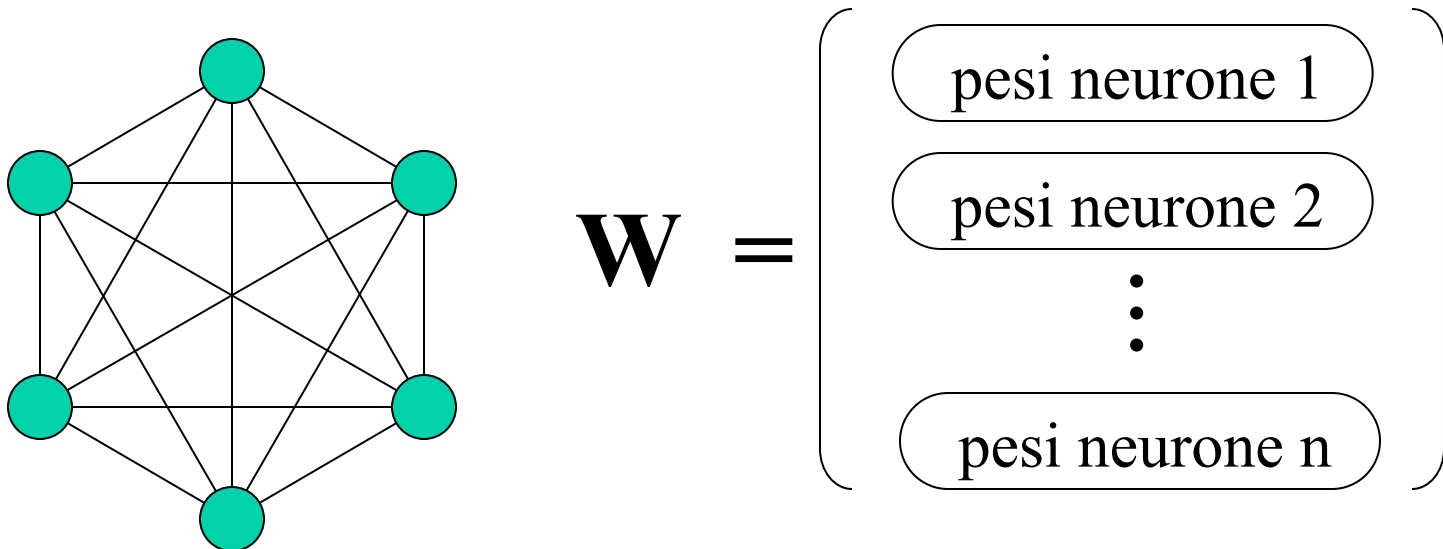


Peso sul neurone j della
connessione proveniente
dal neurone i

Reti completamente connesse

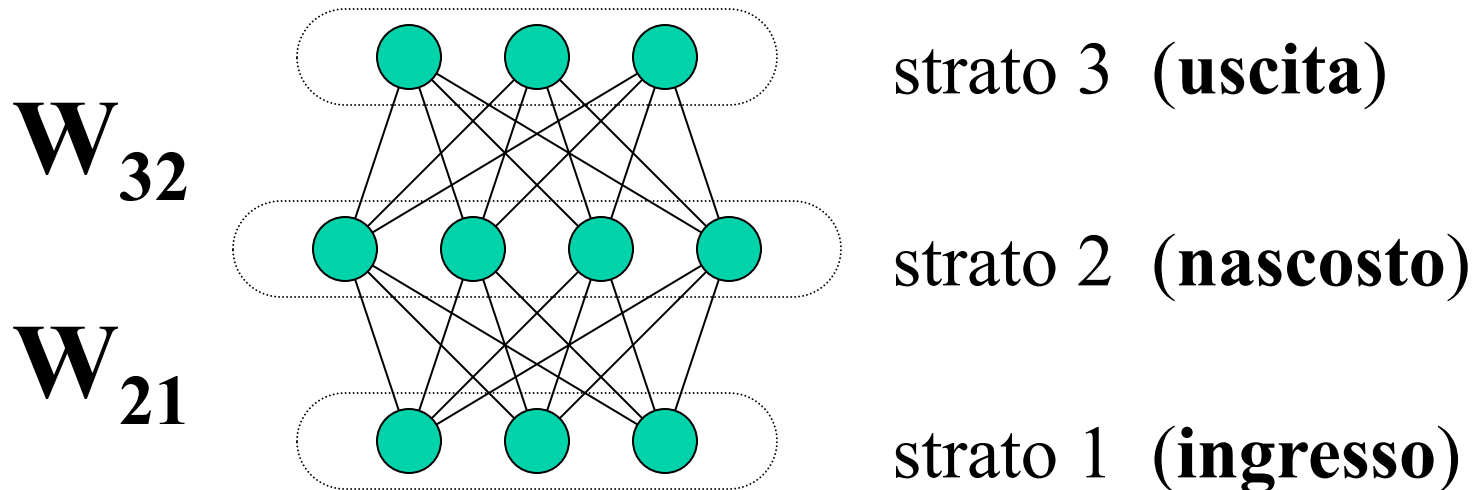
Rappresentano stati che evolvono nel tempo

I pesi della rete possono essere specificati attraverso una **matrice di connessione**



Reti stratificate

I pesi di una rete a **n** strati possono essere specificati attraverso **n-1** **matrici di connessione**:



Modalità di attivazione

- **Sincrona (parallela)**

I neuroni cambiano stato tutti insieme, sincronizzati da un clock.

- **Asincrona (sequenziale)**

I neuroni cambiano stato uno per volta.
Occorre definire un criterio di scelta.

Solo le reti completamente connesse hanno entrambi i tipi di attivazione

Apprendimento

Hebbs (neurofisiologo) ha scoperto che l'apprendimento avviene per cambiamento delle sinapsi.

Capacità della rete di modificare il comportamento in una direzione desiderata al variare delle connessioni sinaptiche (pesi).

I paradigmi di apprendimento possono essere suddivisi in tre classi fondamentali:

- **supervisionato**
- **competitivo**
- **con rinforzo**

Apprendimento supervisionato

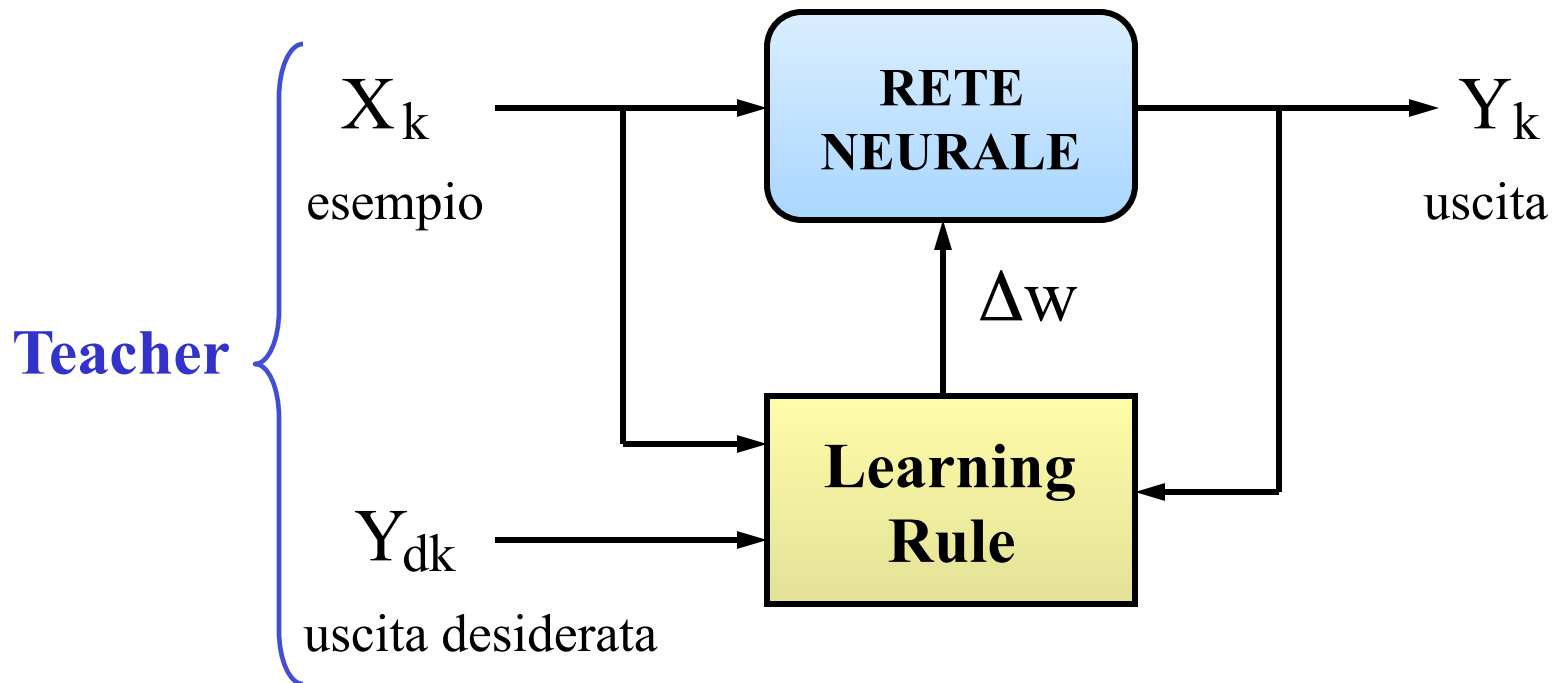
E' il piu' utilizzato

La rete impara a riconoscere un insieme di configurazioni di ingresso desiderate.

La rete opera in due fasi distinte:

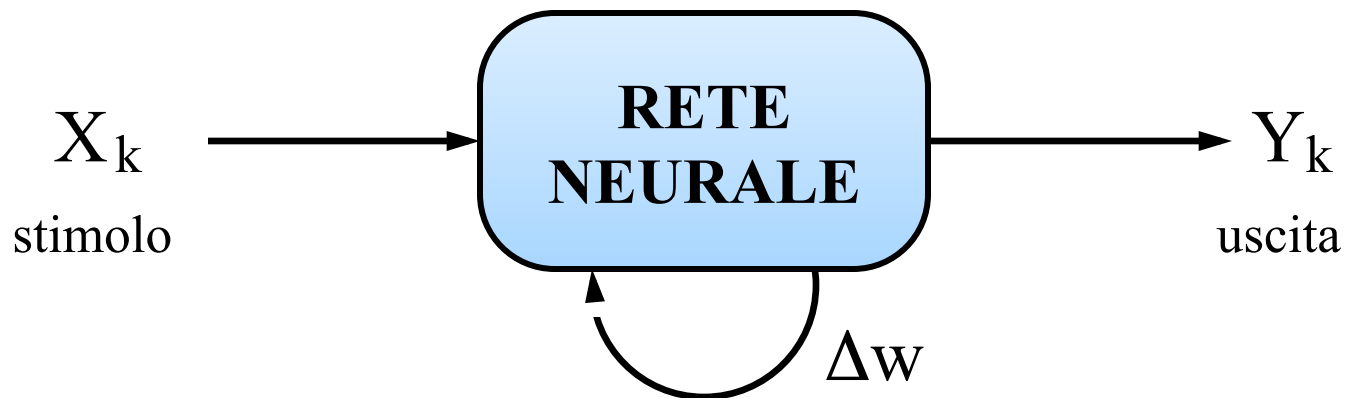
- **Fase di addestramento**
si memorizzano le informazioni desiderate tramite esempi
- **Fase di evoluzione**
si recuperano le informazioni memorizzate

Fase di addestramento



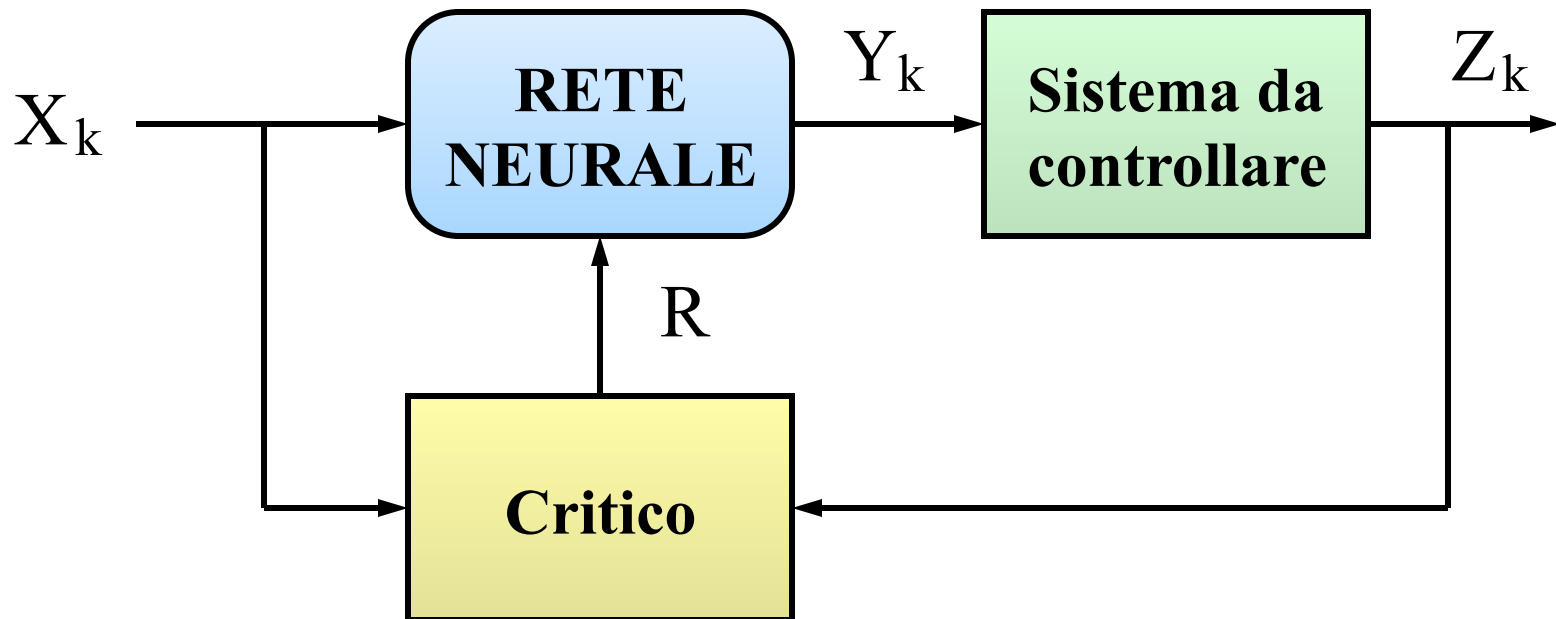
Apprendimento competitivo

- I neuroni competono per specializzarsi a riconoscere un particolare stimolo. Stimoli simili finiscono nella stessa classe.
- Alla fine, ogni stimolo attiva un particolare neurone (isomorfismo tra stimoli e neuroni di uscita).



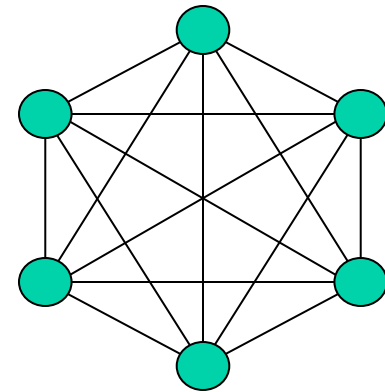
Apprendimento con rinforzo

- Simula l'apprendimento negli animali basato su premi e punizioni: applicazioni per sistemi di controllo



Rete completamente connessa

- Neuroni binari a soglia
- Attivazione parallela



Transizione di stato

$$x_i(t+1) = \text{HS} [\sum_i w_i x_i(t)]$$

$$x_i(t+1) = \begin{cases} 1 & \text{se } \sum_i w_i x_i(t) \geq 0 \\ 0 & \text{altrimenti} \end{cases}$$

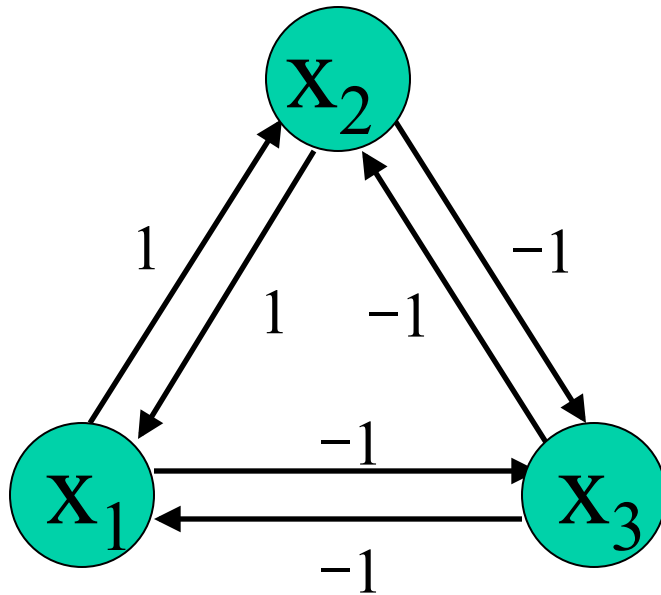
Equazione di evoluzione

In forma matriciale:

$$\mathbf{X}(t+1) = \mathbf{HS} [\mathbf{W} \mathbf{X}(t)]$$

- $\mathbf{X}(t)$ è lo stato della rete al tempo t
- \mathbf{W} è la matrice dei pesi

Esempio



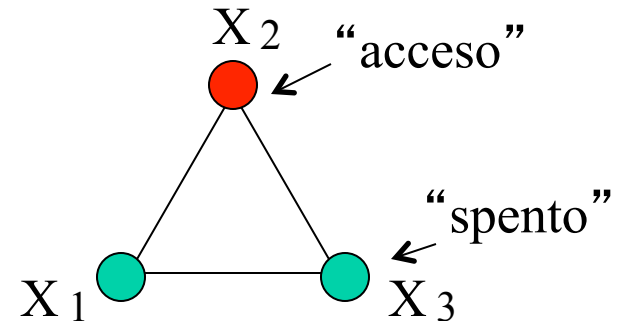
Matrice simmetrica

$$\mathbf{W} = \begin{pmatrix} 0 & 1 & -1 \\ 1 & 0 & -1 \\ -1 & -1 & 0 \end{pmatrix}$$

Mancano le
connessioni verso se
stessi: diagonale a 0

Transizione di stato

Stato iniziale: $\mathbf{X}(t) = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$



Stato successivo: $\mathbf{X}(t+1) = \mathbf{HS}[\mathbf{W} \mathbf{X}(t)] =$

$$= \mathbf{HS} \left[\begin{pmatrix} 0 & 1 & -1 \\ 1 & 0 & -1 \\ -1 & -1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \right] = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$$

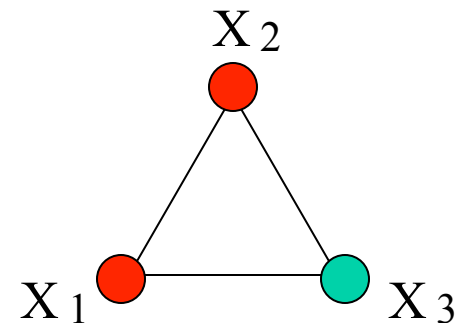
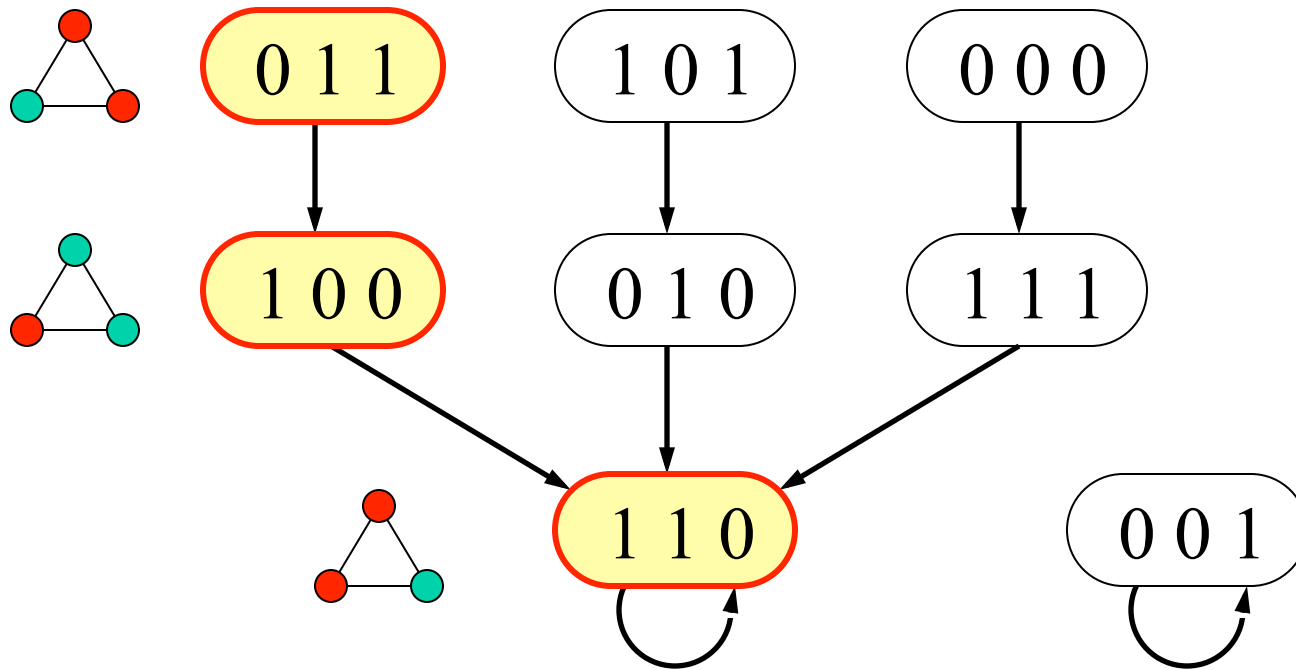


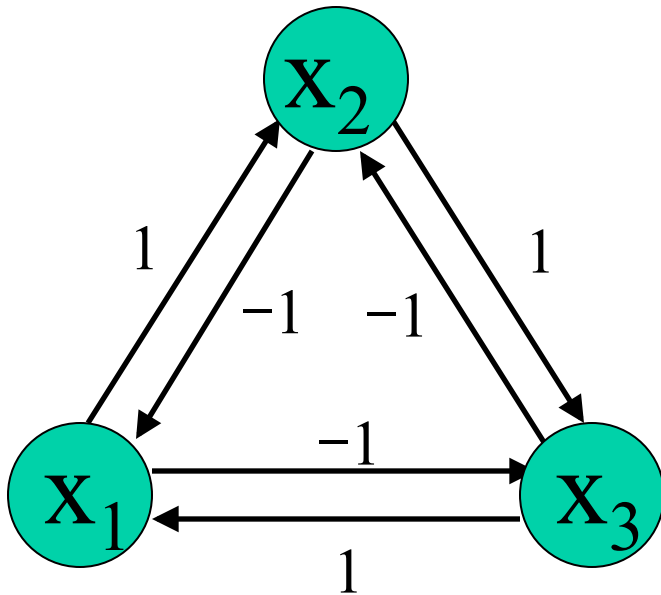
Diagramma delle transizioni



La rete percorre una traiettoria fino a stati stabili

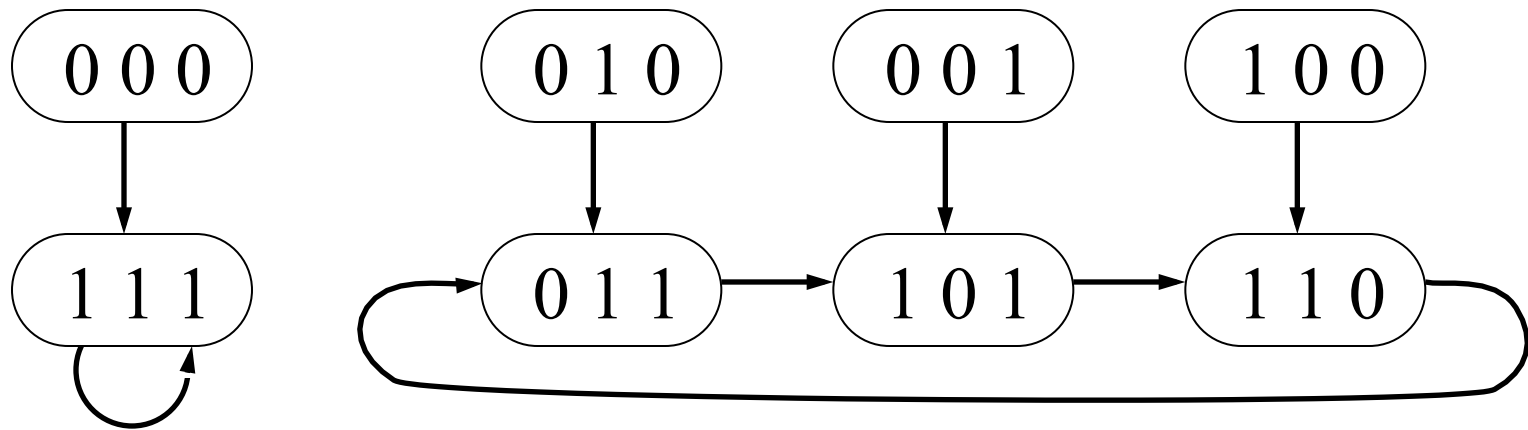
Esempio

Matrice antisimmetrica



$$\mathbf{W} = \begin{pmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{pmatrix}$$

Diagramma delle transizioni



Definizioni

- **Trasformazione**

Funzione $T:S \rightarrow S$, che trasforma uno stato $X(t)$ nel successivo $X(t+1)$.

- **Traiettorie**

Sequenza degli stati assunti dalla rete, a partire da uno stato iniziale X_0 :

$$X(0) = X_0$$

$$X(t+1) = T[X(t)]$$

Definizioni

- **Ciclo limite di ordine k**

Traiettoria che parte da uno stato iniziale X_I e arriva nello stesso stato dopo k passi.

- **Stato stabile**

Stato che genera una traiettoria costante:

$$X(t+1) = X(t) = X_s$$

Definizioni

- **Stato raggiungibile**

Uno stato X_F si dice raggiungibile da X_I se esiste una traiettoria che parte da X_I e arriva in X_F .

- **Stabilità globale**

Una rete si dice globalmente stabile se per ogni stato iniziale X , la traiettoria che parte da X raggiunge uno stato stabile.

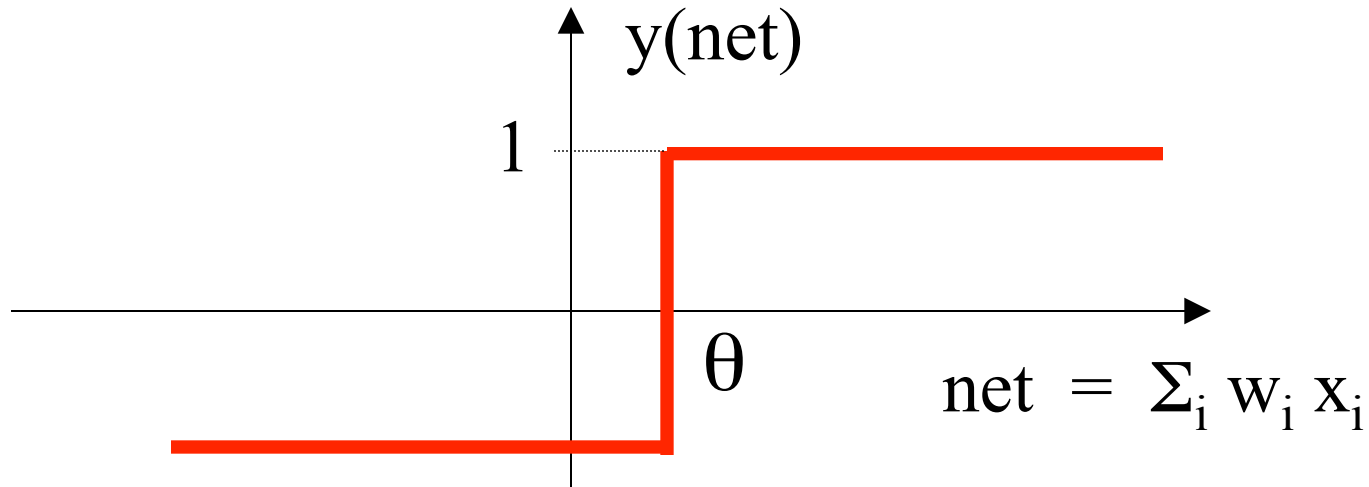
Proprietà di stabilità

(Hopfield '82)

Una rete neurale completamente connessa è globalmente stabile se:

- la matrice dei pesi è simmetrica
- l'attivazione è asincrona

Modello di Hopfield



$$y = \text{sgn}\left(\sum_{i=1}^n w_i x_i - \theta\right)$$

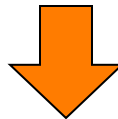
La funzione Energia

- Ogni stato è caratterizzato da una energia:

$$E(\mathbf{X}) = -\frac{1}{2} \mathbf{X}^T \mathbf{W} \mathbf{X}$$

- Se la matrice dei pesi è simmetrica e l'attivazione è asincrona, allora


**$E(\mathbf{X})$ è monotona non crescente
con l'evolvere dello stato**



$$E[\mathbf{X}(t+1)] \leq E[\mathbf{X}(t)]$$

Dimostrazione

Con questa ipotesi posso
derivare rispetto a una
singola variabile

$$E(X) = -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n w_{ij} x_i x_j$$


Derivando rispetto a x_k (varia un solo neurone):

$$\frac{\partial E}{\partial x_k} = -\frac{1}{2} \left(\sum_{j=1}^n w_{kj} x_j + \sum_{i=1}^n w_{ik} x_i \right)$$

Supponendo W simmetrica si ha:

$$\frac{\partial E}{\partial x_k} = - \left(\sum_{j=1}^n w_{kj} x_j \right)$$

Nel discreto si ha:

$$\Delta E = -\Delta x_i \left(\sum_{j=1}^n w_{ij} x_j \right)$$

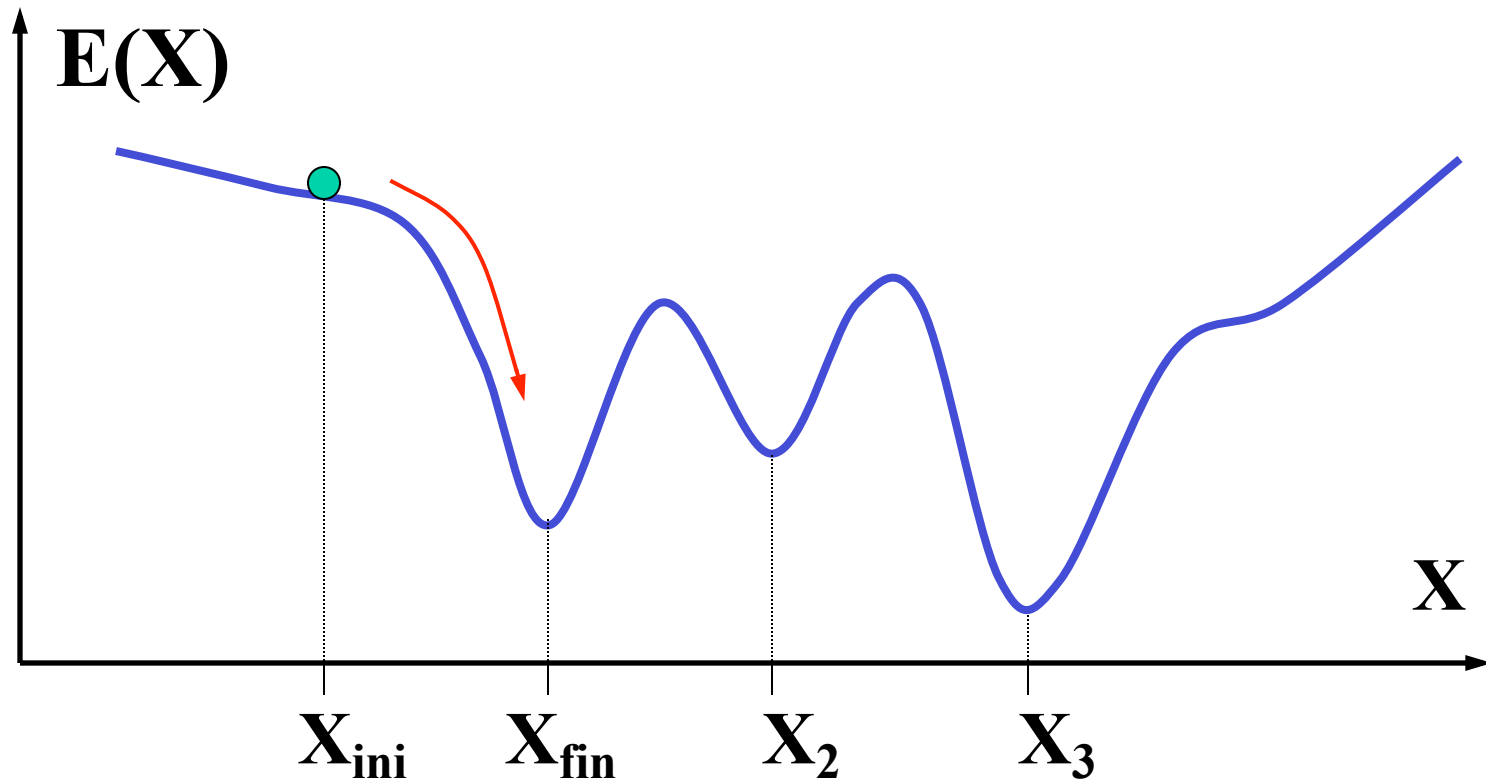
$$\Delta x_i > 0 \quad (x_i: -1 \rightarrow 1) \quad \Rightarrow \quad \sum_j w_{ij} x_j \geq 0$$

$$\Delta x_i < 0 \quad (x_i: 1 \rightarrow -1) \quad \Rightarrow \quad \sum_j w_{ij} x_j < 0$$

quindi:

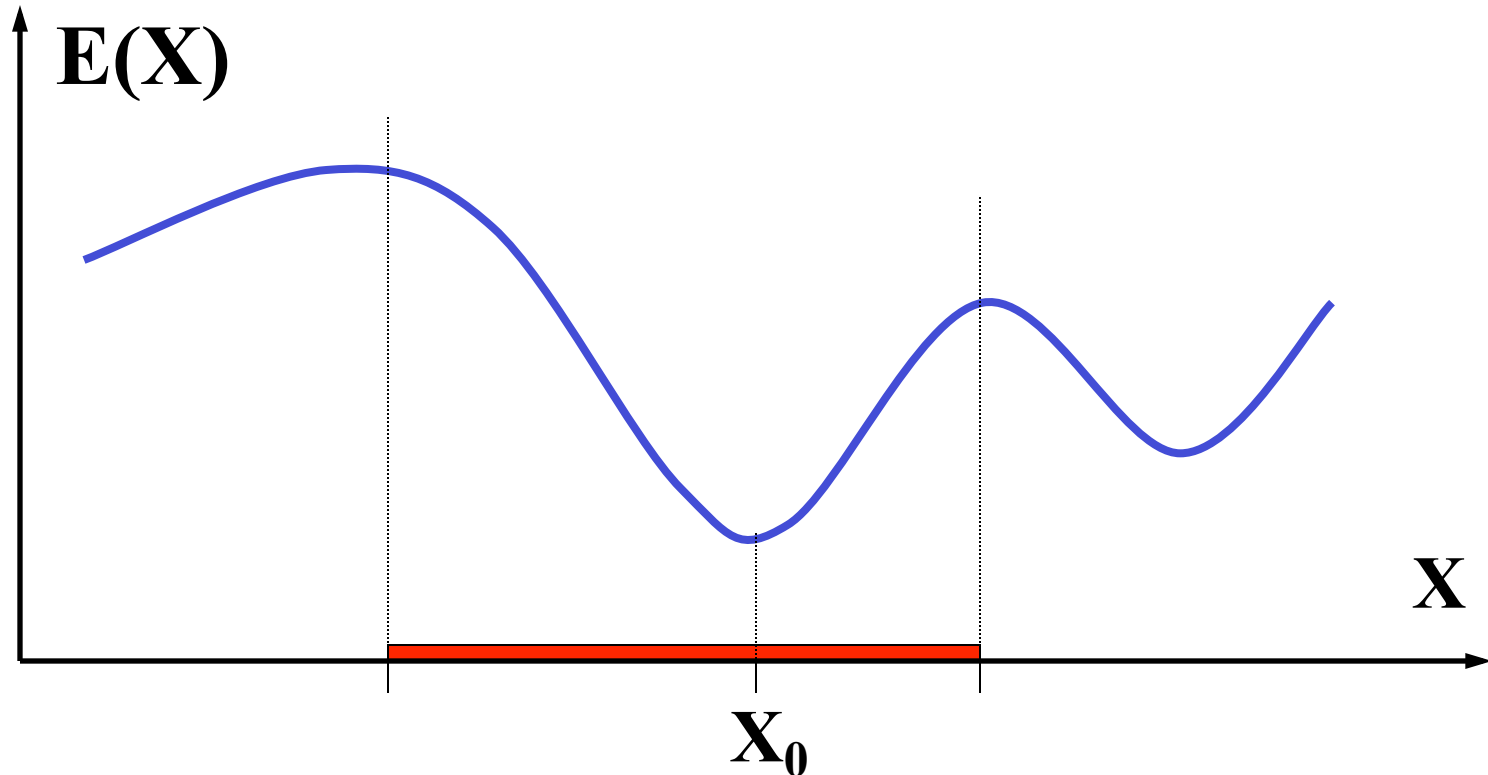
$$\Delta E < 0$$

La rete evolve verso uno stato stabile

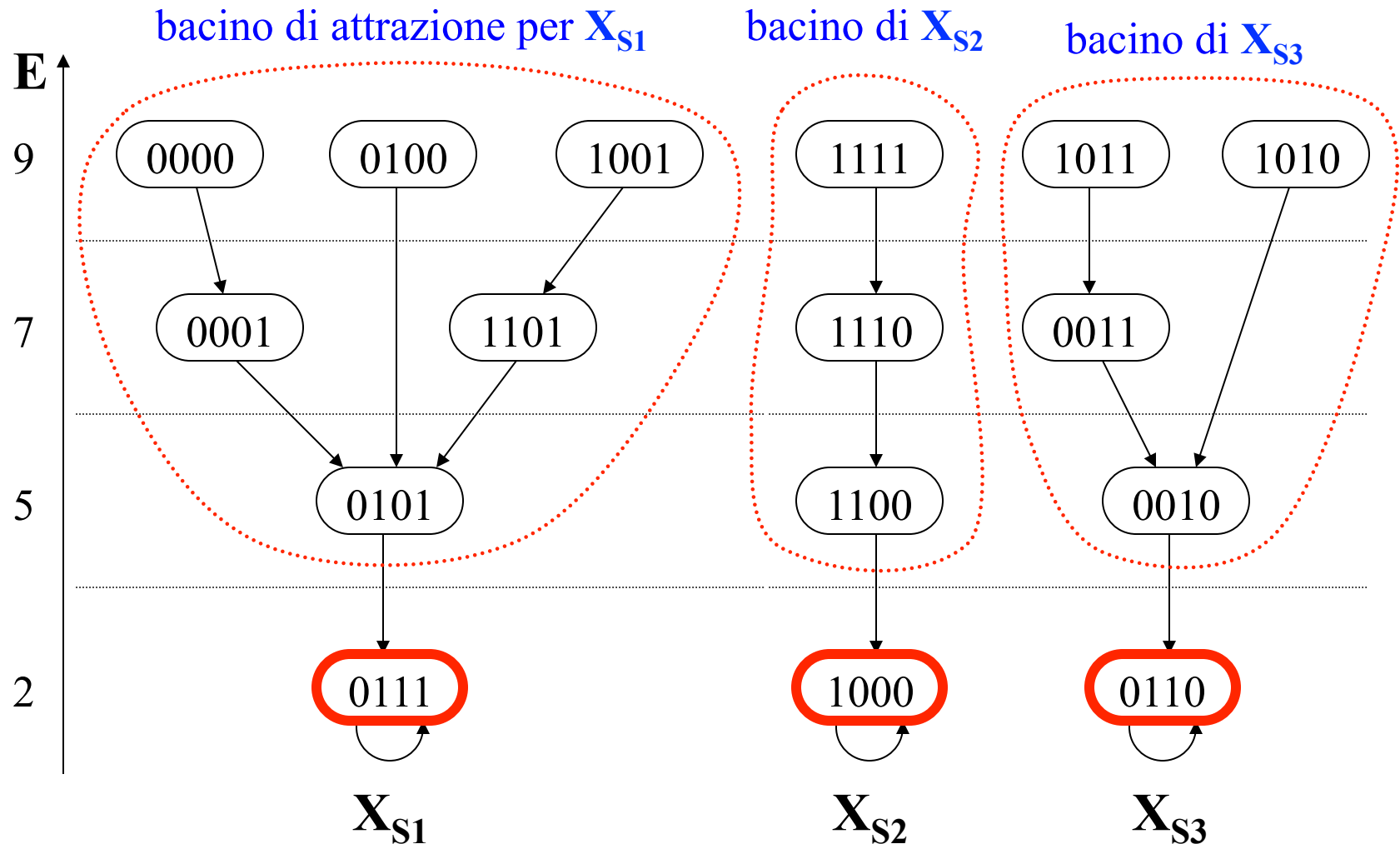


Bacino di attrazione:

insieme degli stati tali che tutte le traiettorie che partono da essi finiscono nello stesso stato stabile.

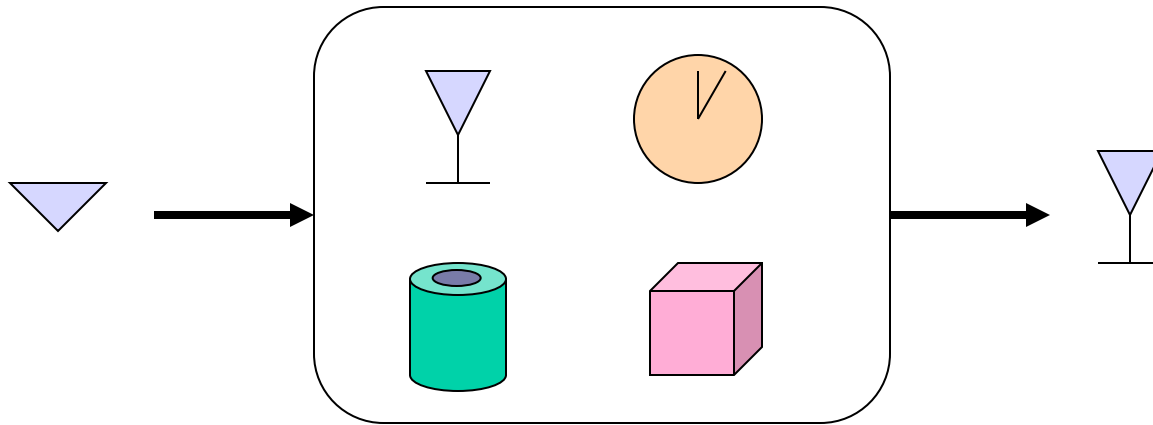


Rete con 3 stati stabili



Memorie Associative

Sono memorie i cui i contenuti possono essere recuperati sulla base di una informazione **parziale** o **distorta** del contenuto stesso.



Come facciamo a creare buchi energetici nelle posizioni giuste?

Memorizzazione di immagini

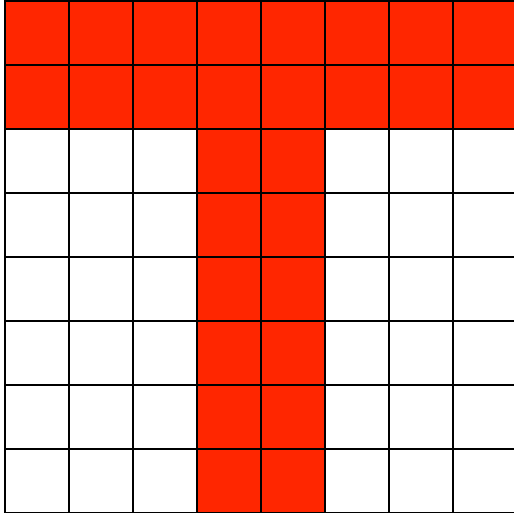


Immagine: $n \times m$ pixel

Neuroni: $N = n \times m$

Connessioni: $C = N^2$

Stati: $S = 2^N$

Immagine: 8×8 pixel

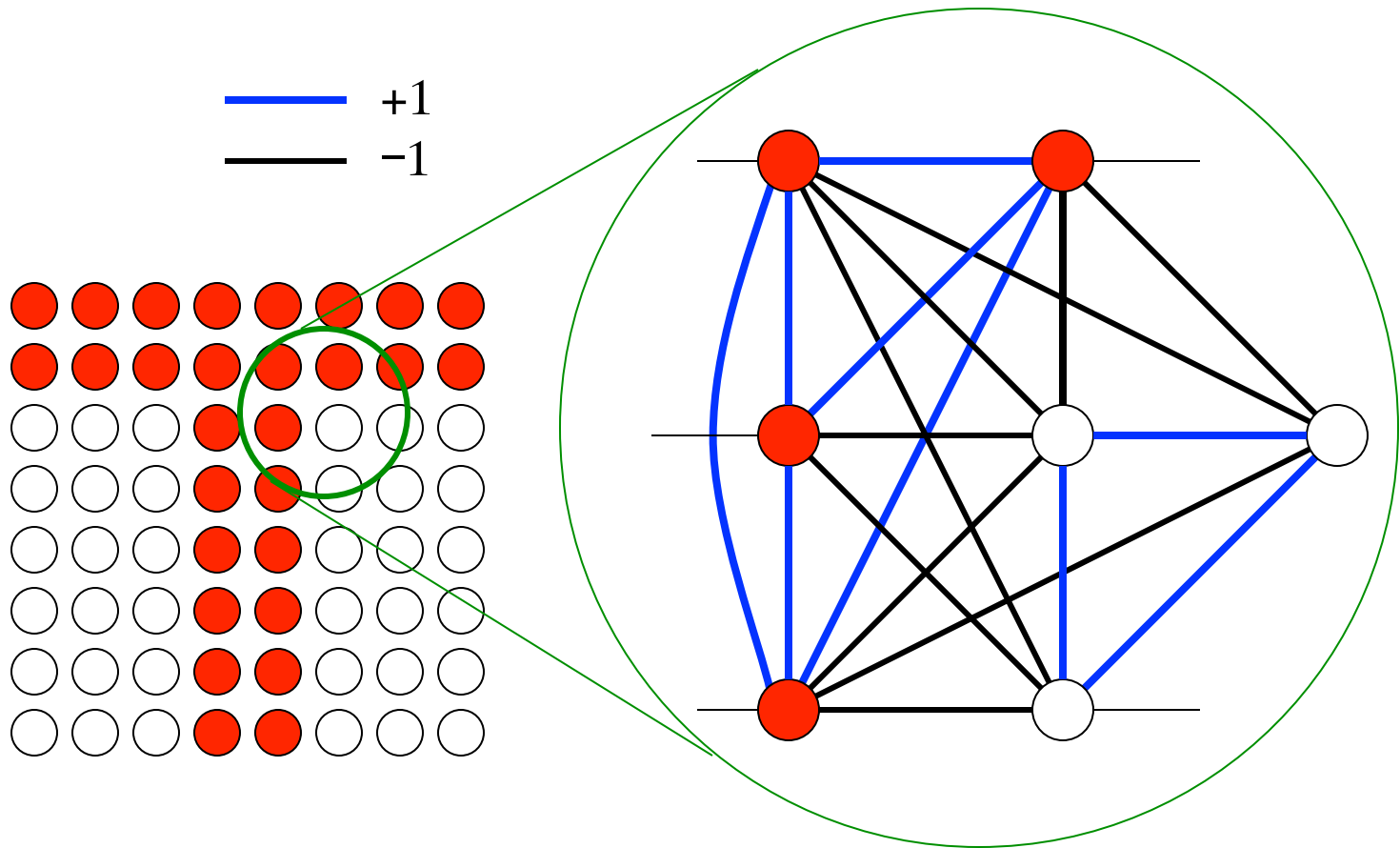
Neuroni: $N = 64$

Connessioni: $C = 4096$

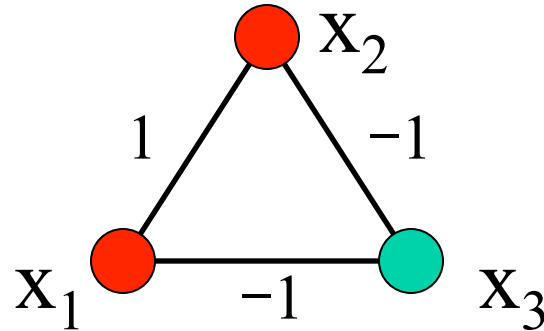
Stati: $S \cong 2 \cdot 10^{19}$

Regola di memorizzazione

(Hopfield '82)

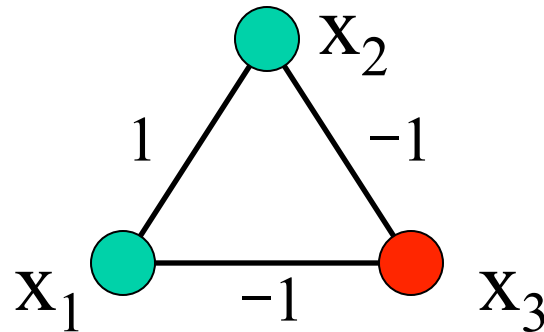


M1: (+ + -)



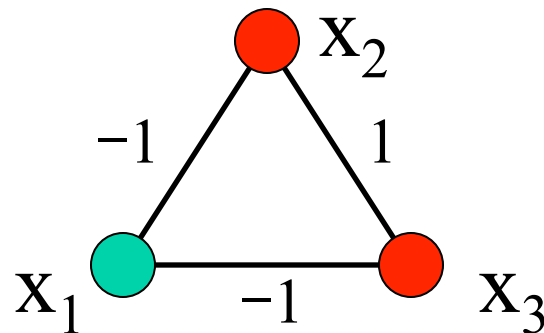
$$W_1 = \begin{bmatrix} 0 & 1 & -1 \\ 1 & 0 & -1 \\ -1 & -1 & 0 \end{bmatrix}$$

M2: (- - +)



$$W_2 = \begin{bmatrix} 0 & 1 & -1 \\ 1 & 0 & -1 \\ -1 & -1 & 0 \end{bmatrix}$$

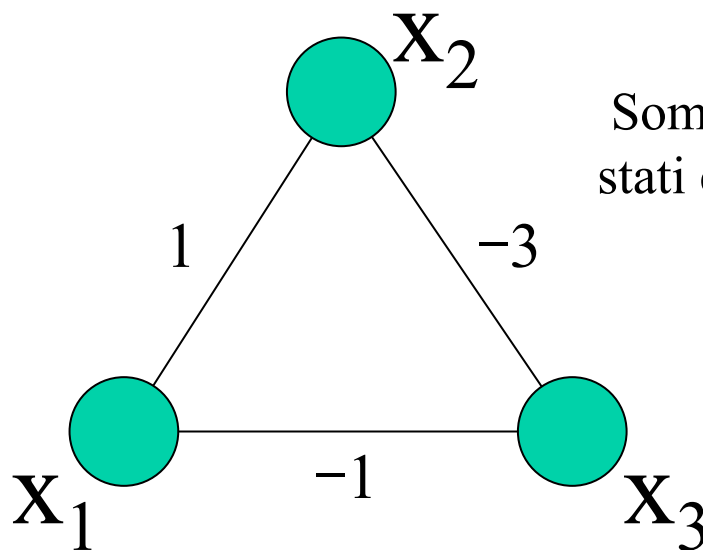
M3: (- + +)



$$W_3 = \begin{bmatrix} 0 & -1 & -1 \\ -1 & 0 & 1 \\ -1 & 1 & 0 \end{bmatrix}$$

Rete complessiva

$$W = \sum_{k=1}^m W_k = \begin{pmatrix} 0 & 1 & -3 \\ 1 & 0 & -1 \\ -3 & -1 & 0 \end{pmatrix}$$



Sommo le matrici relative ai singoli
stati che voglio ottenere come stabili
FUNZIONA???

Diagramma delle transizioni

(Attivazione Sincrona)

$$M = \{(+ + -), (- - +), (- + +)\}$$

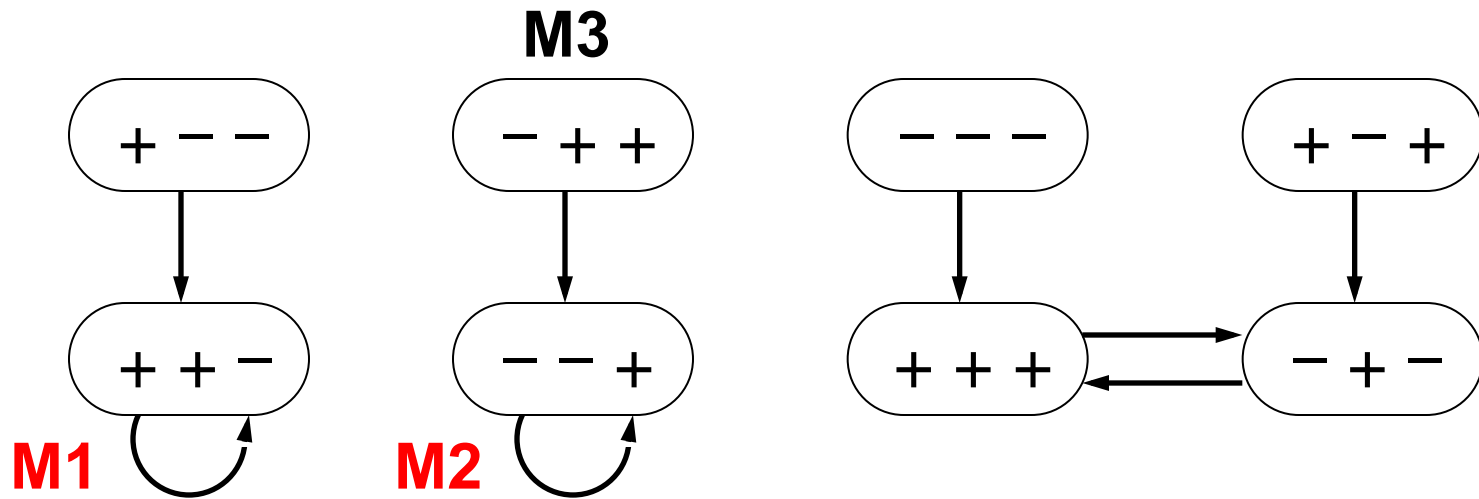
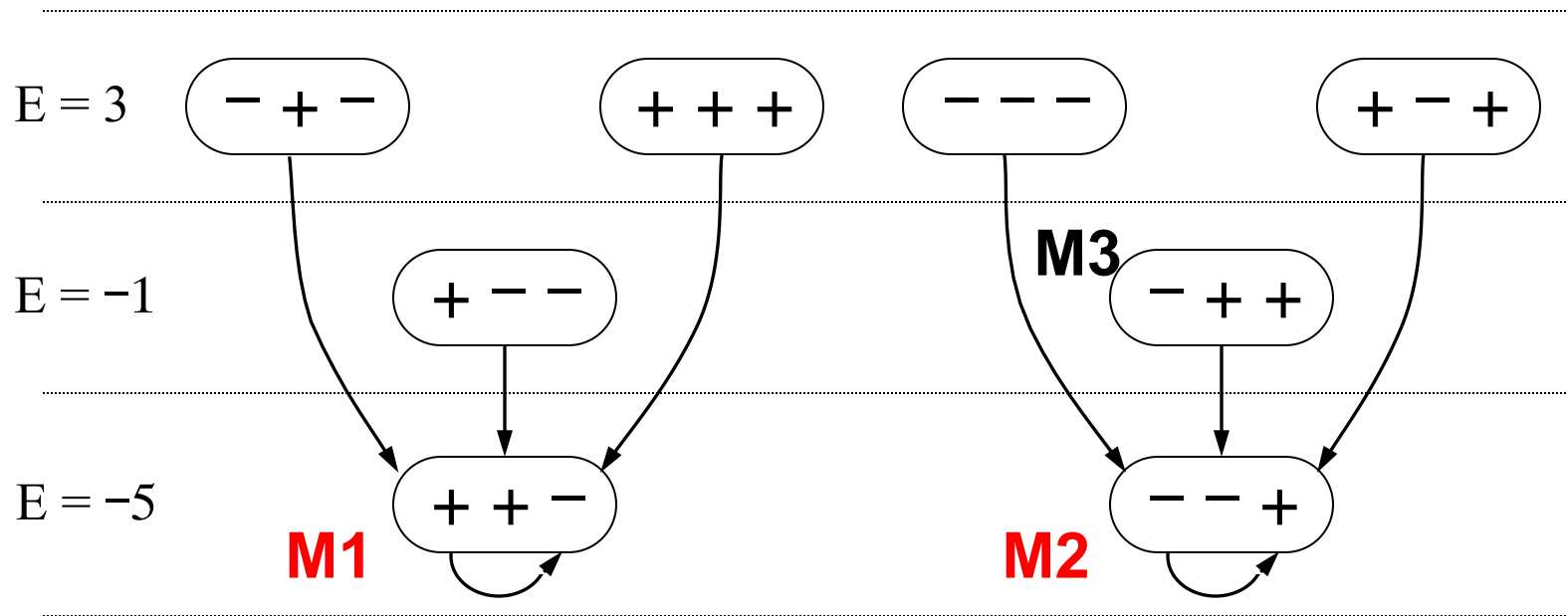


Diagramma delle transizioni (Attivazione Asincrona)

$$M = \{(+ + -), (- - +), (- + +)\}$$



Osservazioni

Quando si sovrappongono troppe memorie:

- **Non sempre una memoria risulta stabile**

La creazione di un minimo locale può avere l'effetto di cancellarne un altro.

- **Possono nascere memorie spurie**

La superficie energetica può assumere forme complesse.

Capacità di memoria

- **Regola empirica (Hopfield)**

Una rete di N neuroni può ospitare al più un numero $M = N/7$ memorie ($M \cong 0.15 N$).

- **Analisi statistica**

Detta β la probabilità di stabilità delle memorie,

$$M = \frac{N}{2 \ln(N/a)} \quad \text{dove} \quad a = -\ln \beta$$

Ad esempio, se $N = 1000$ e $\beta = 0.9$ si ha $M = 54$.

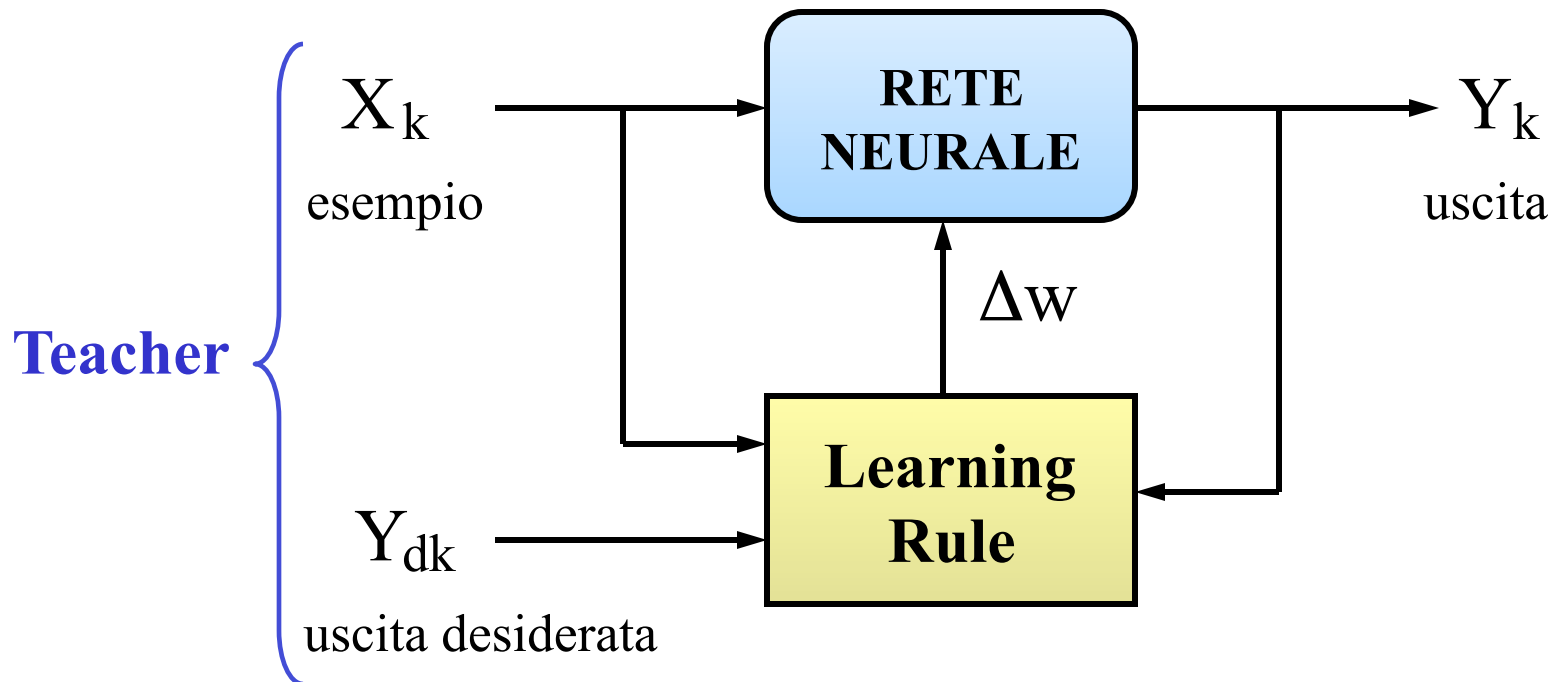
Apprendimento con supervisione

La rete impara ad associare un insieme di coppie (X_k, Y_{dk}) desiderate.

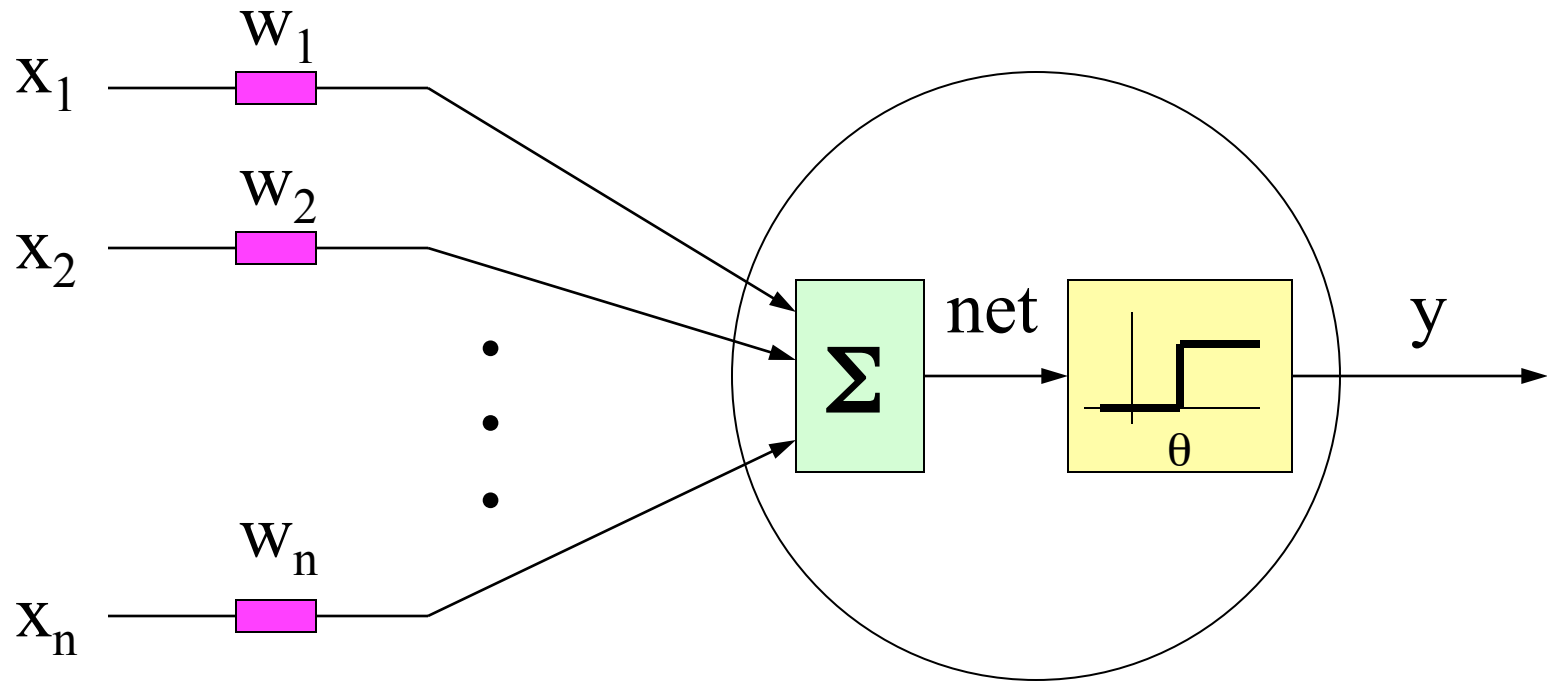
La rete opera in due fasi distinte:

- **Fase di addestramento**
si memorizzano le informazioni desiderate
- **Fase di evoluzione**
si recuperano le informazioni memorizzate

Fase di Addestramento



II Perceptron (Rosenblatt ' 58)

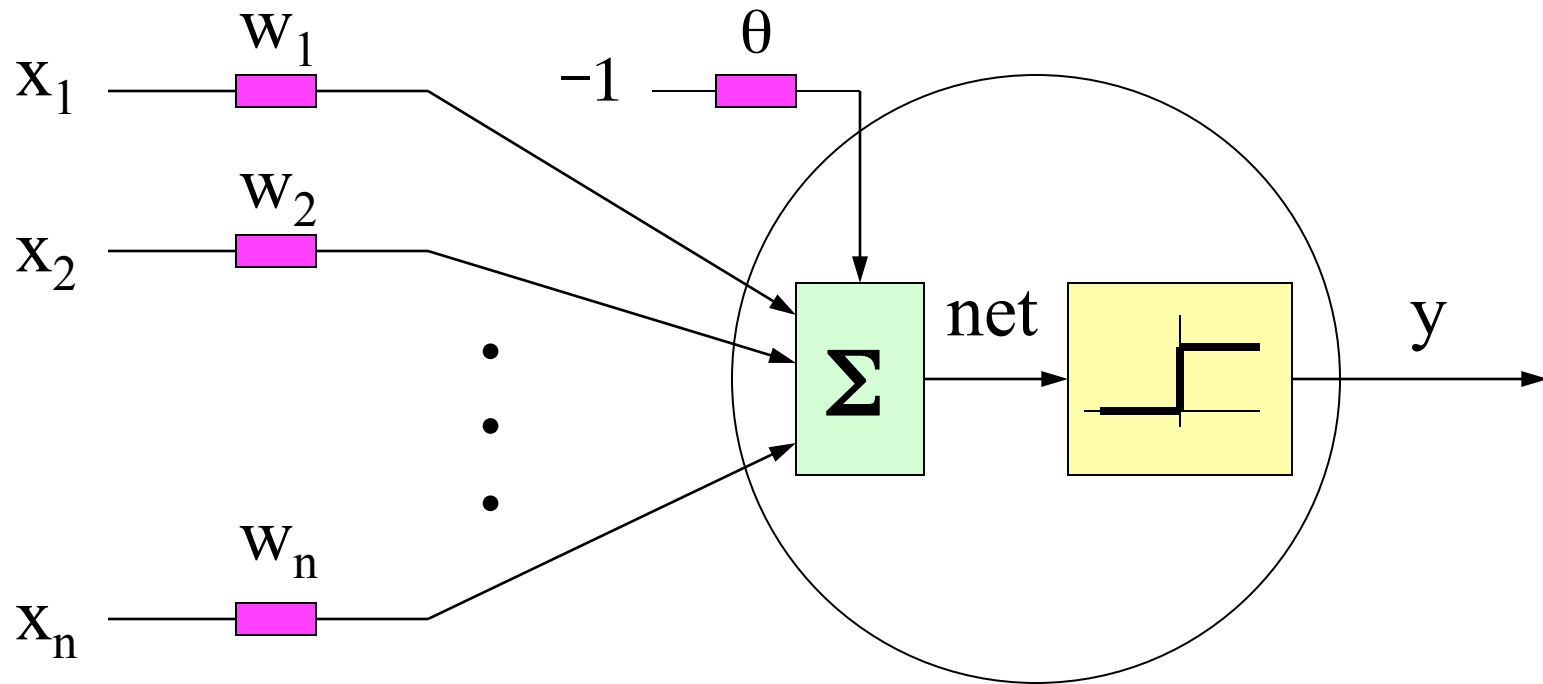


Ingressi
binari

$$\text{net} = \sum_i w_i x_i$$
$$y = \text{HS}(\text{net} - \theta)$$

Uscita
binaria

II Perceptron (Rosenblatt ' 58)



Ingressi
binari

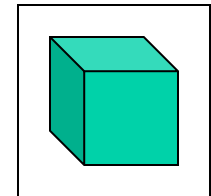
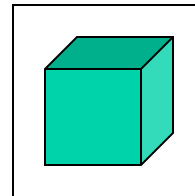
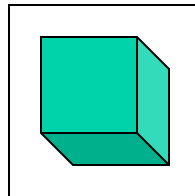
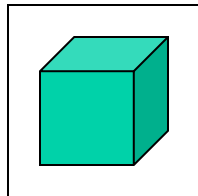
$$\begin{aligned} \text{net} &= \sum_i w_i x_i - \theta \\ y &= \text{HS}(\text{net}) \end{aligned}$$

Uscita
binaria

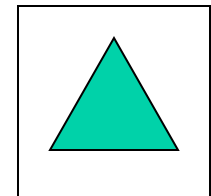
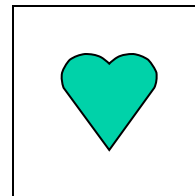
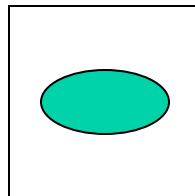
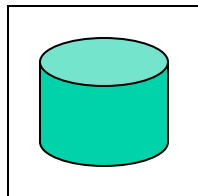
Classificazione

- Un perceptron può essere addestrato a riconoscere se un pattern d'ingresso X appartenga o no ad una classe C :

CUBO
(1)



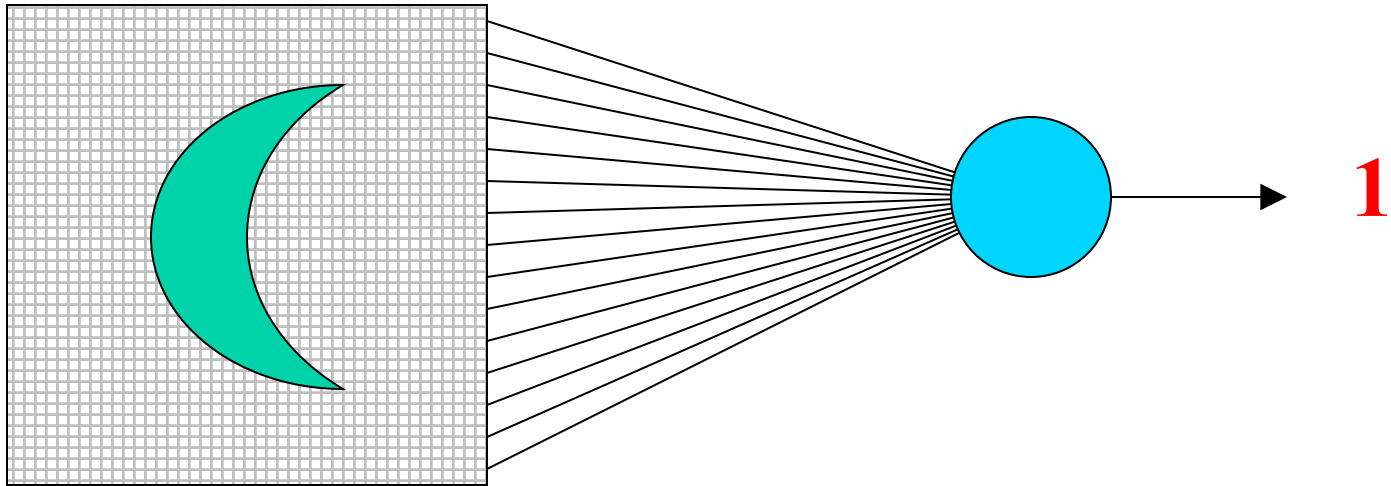
NON
CUBO
(0)



L' esperimento di Rosenblatt

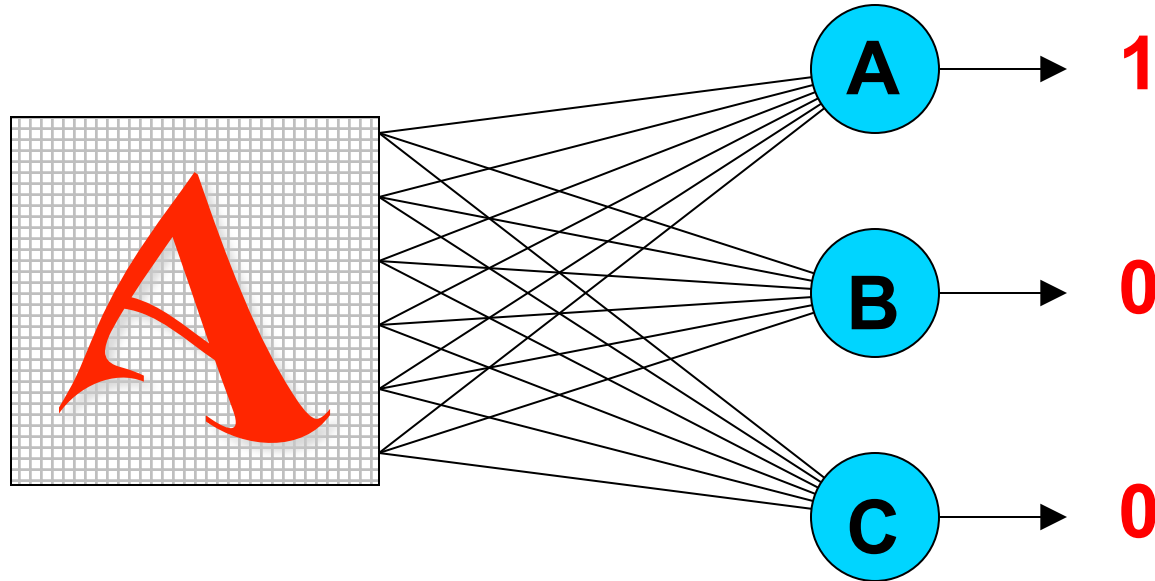
1 = concavo

0 = convesso

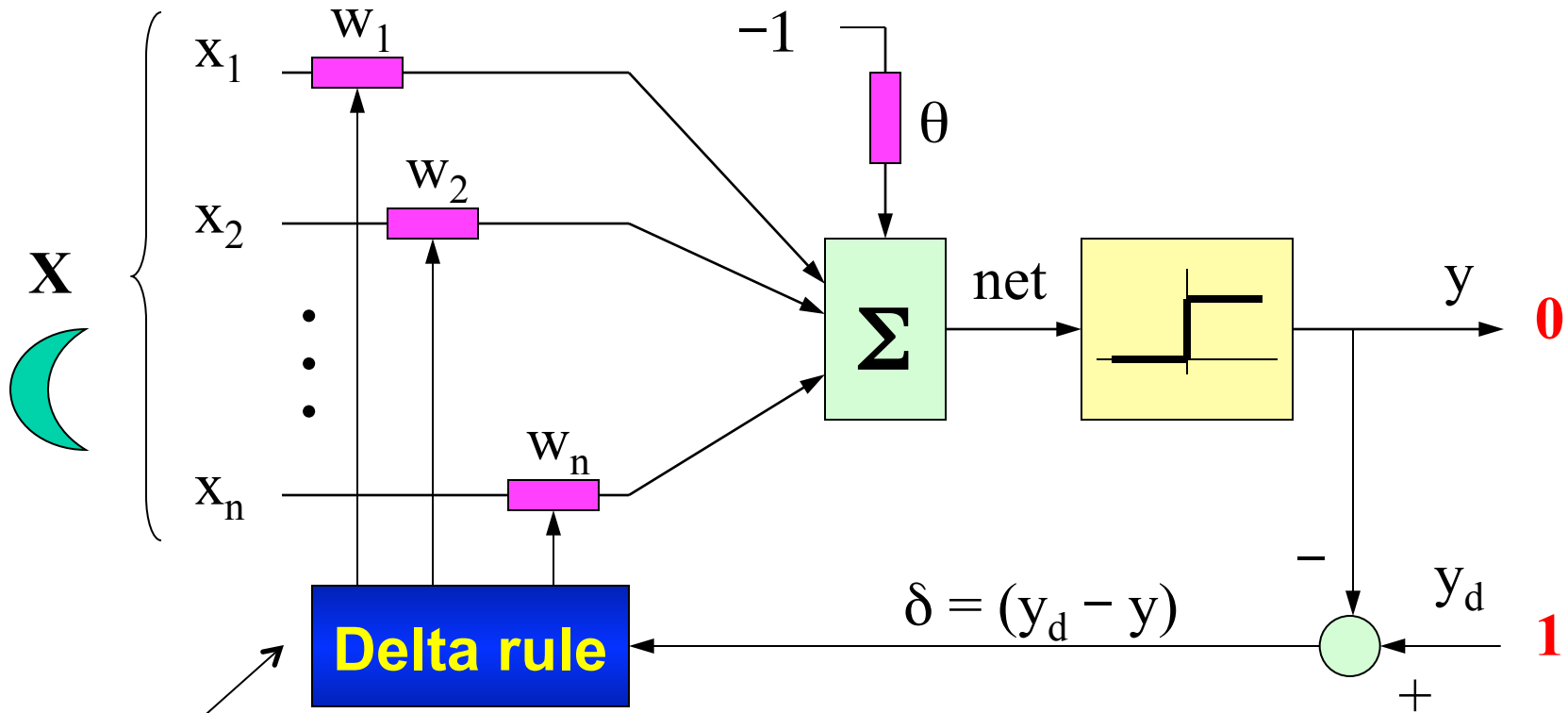


Reti di Perceptron

Riconoscitore di caratteri



Addestramento



θ incorporato nei pesi.
cambiando i pesi
apprendo θ

$$w_i(t+1) = w_i(t) + \eta \delta x_i$$

η = coefficiente di apprendimento (learning rate)

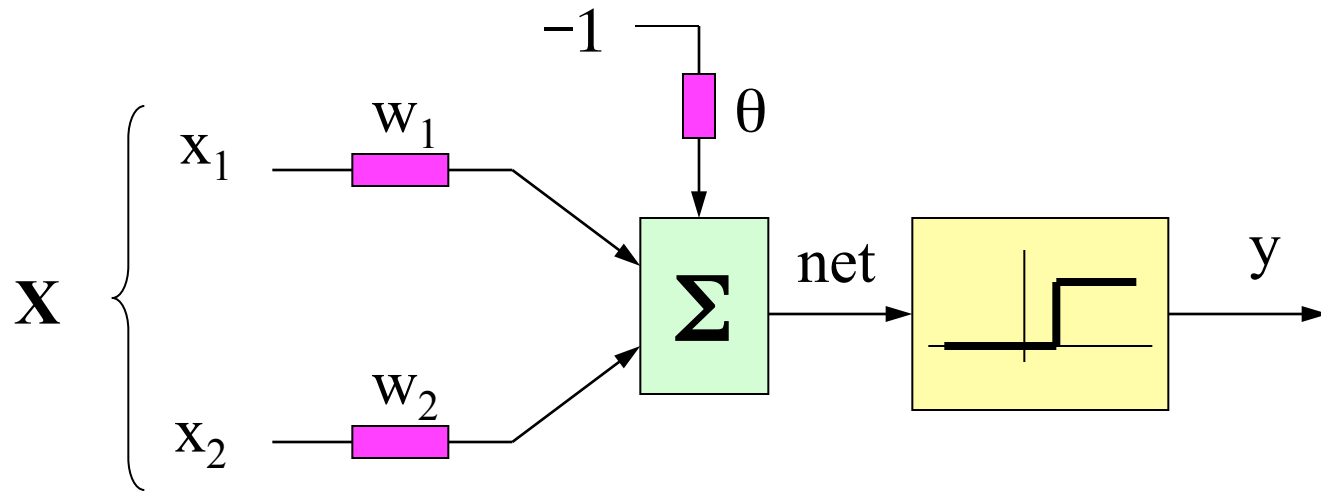
Algoritmo di apprendimento

1. Si fissa un training set di M esempi:

$$TS = \{(X_k, y_{dk}), k = 1, M\}$$

2. si inizializzano i pesi con valori casuali w_i ;
3. si presenta una coppia (X_k, y_{dk}) ;
4. si calcola la risposta y_k della rete;
5. si aggiornano i pesi con la *delta rule*: ($\Delta w = \eta \delta x$)
6. si ripete il ciclo dal passo 3, finchè tutte le risposte non siano giuste: $y_k = y_{dk} \quad \forall k \in [1, M]$

Perceptron a due ingressi



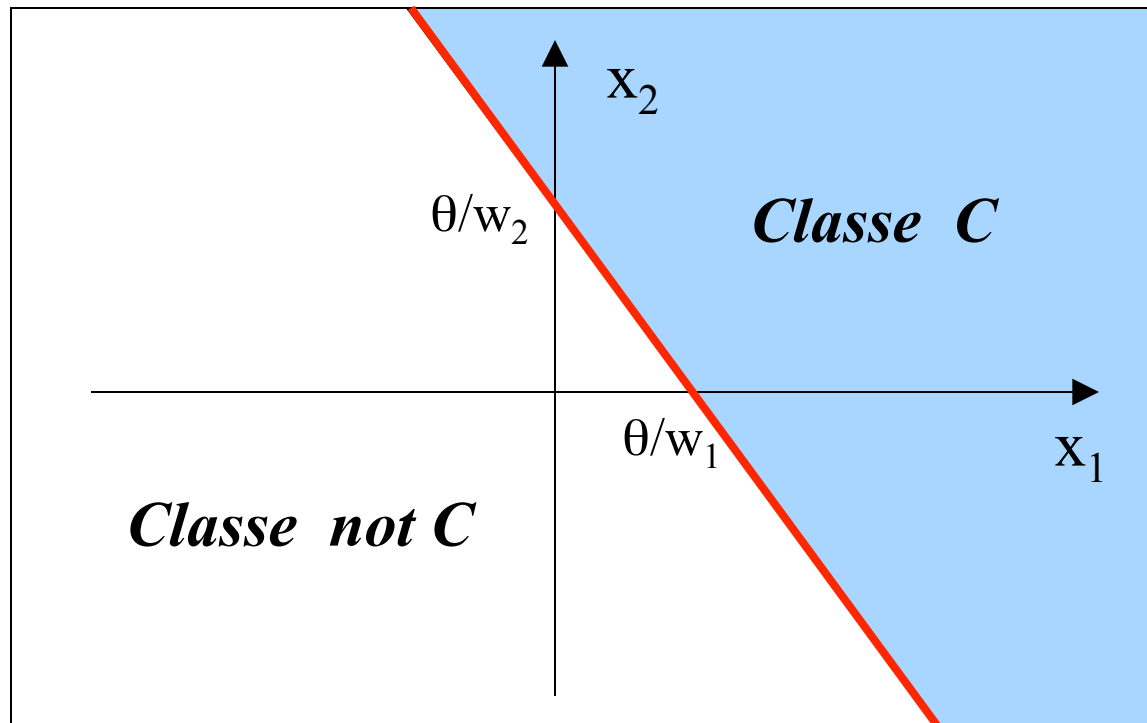
$$y = \text{HS}(w_1 x_1 + w_2 x_2 - \theta)$$

I pattern riconosciuti come appartenenti alla classe saranno quelli per cui:

$$w_1 x_1 + w_2 x_2 - \theta > 0$$

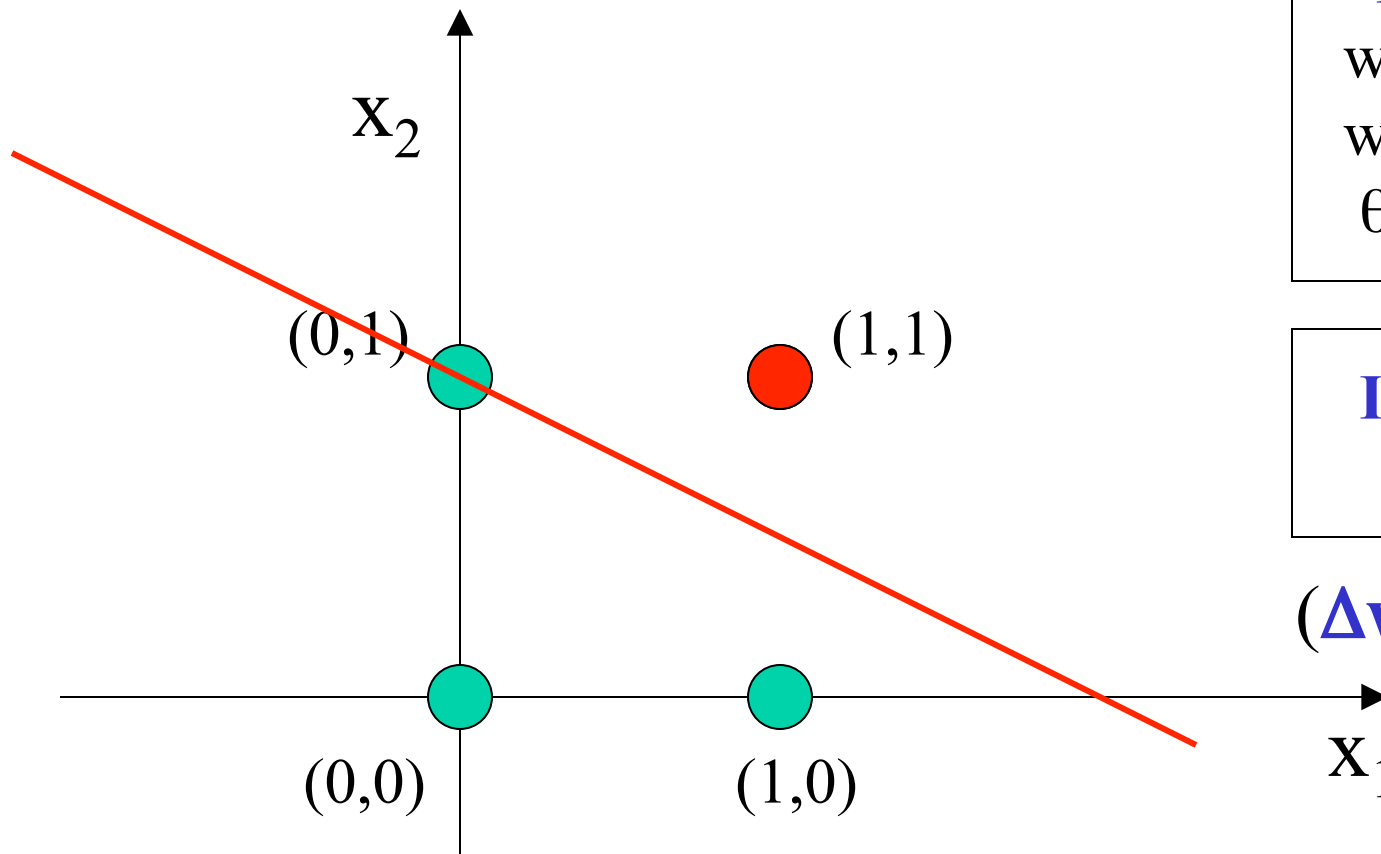
Separazione lineare dello spazio d'ingresso

$$x_2 > -(w_1/w_2) x_1 + \theta / w_2$$



Apprendimento di un AND

$$y = 1 \text{ if } x_2 > -(w_1/w_2) x_1 + \theta / w_2$$



PESI

$$w_1 = 0.5$$

$$w_2 = 1$$

$$\theta = 1$$

INPUT

$$(\Delta w = \eta \delta x)$$

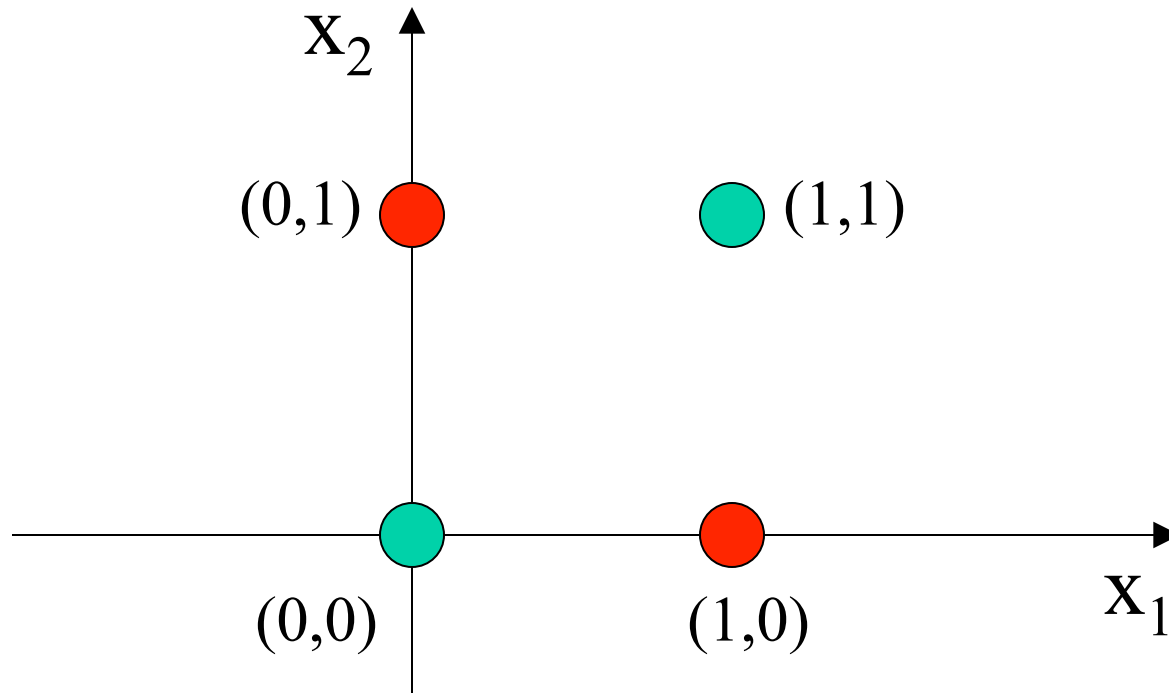
Limiti del Perceptron

Per apprendere una classificazione, il problema deve essere linearmente separabile:

- i pattern appartenenti alla classe C devono essere contenuti in un semipiano dello spazio d'ingresso
- Con n ingressi, lo spazio d'ingresso diventa n -dimensionale e i pattern vengono separati da un iperpiano.

Il problema dello XOR

Non è separabile linearmente!



Possibili soluzioni

- Utilizzare neuroni con funzioni di uscita opportune.
- Combinare la risposta di più neuroni, secondo architetture multistrato.

Funzioni di uscita opportune

- $f(\text{net}) = \text{net}^2, \quad w_1 = 1, w_2 = -1$

$$y = (x_1 - x_2)^2$$

- $f(\text{net}) = |\text{net}|, \quad w_1 = 1, w_2 = -1$

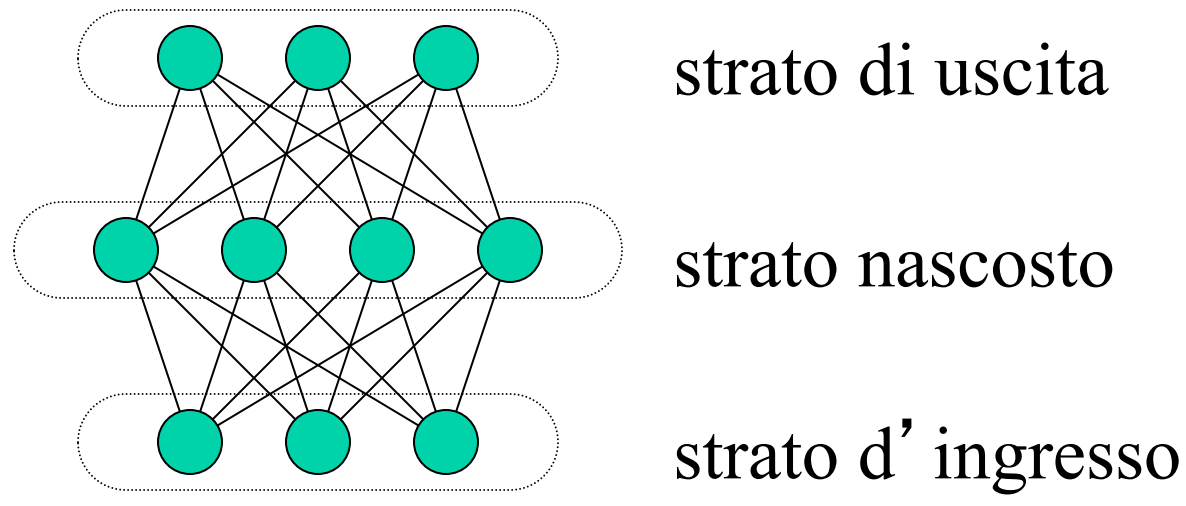
$$y = |x_1 - x_2|$$

- $f(\text{net}) = 1 - e^{-|\text{net}|}, \quad w_1 = 1, w_2 = -1$

$$y = 1 - e^{-|x_1 - x_2|}$$

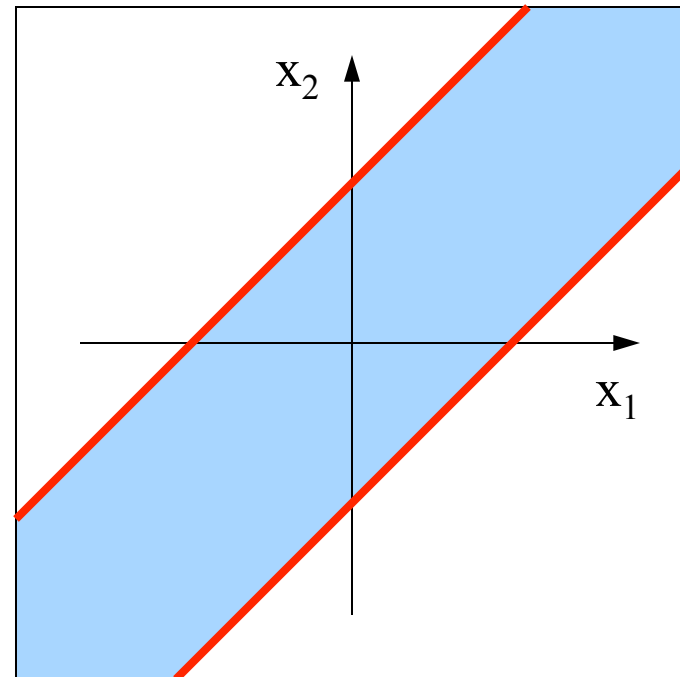
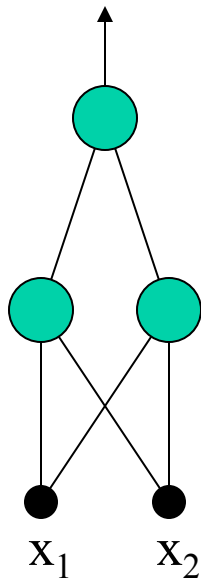
Reti multistrato

- Tutti i neuroni di uno strato sono connessi con tutti i neuroni dello strato successivo.
- Non esistono connessioni tra neuroni dello stesso strato.



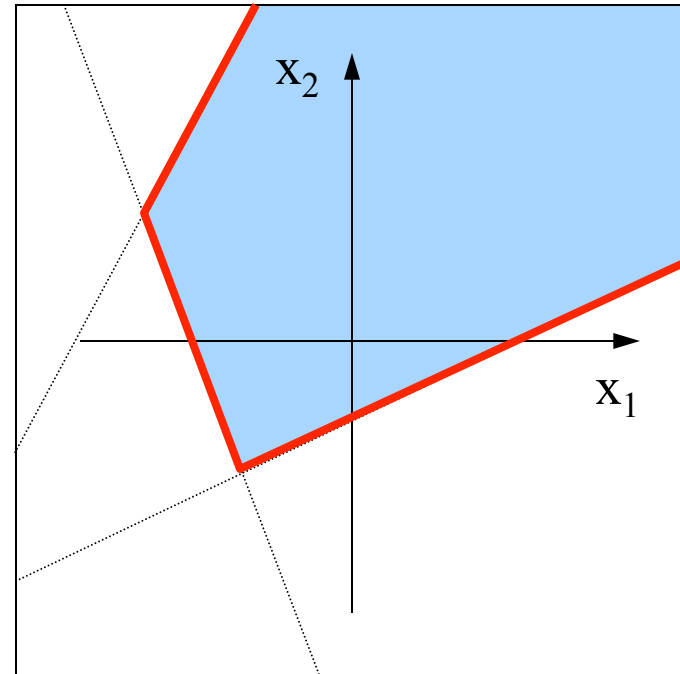
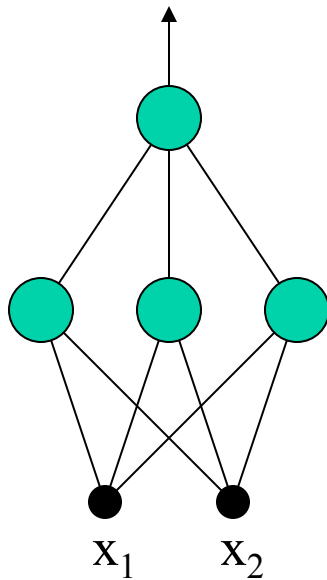
Reti a tre strati

- Sono in grado di separare regioni convesse numero di lati \leq numero neuroni nascosti



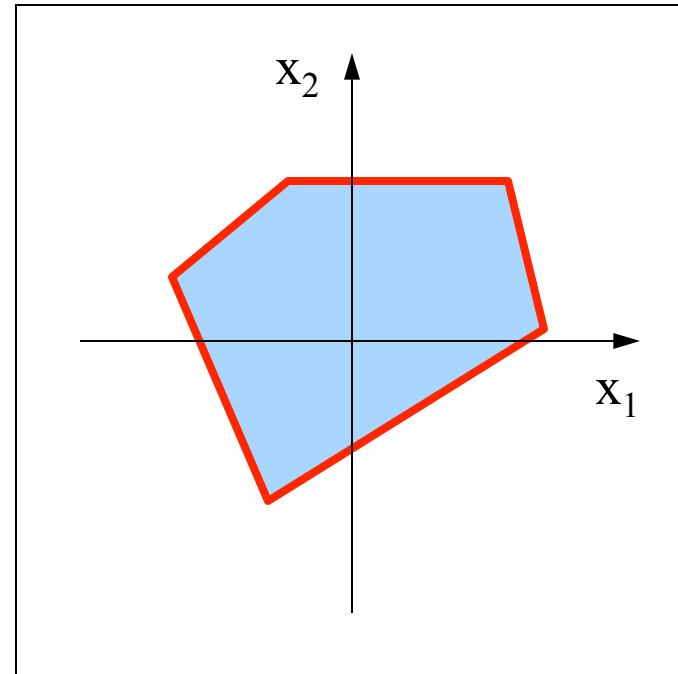
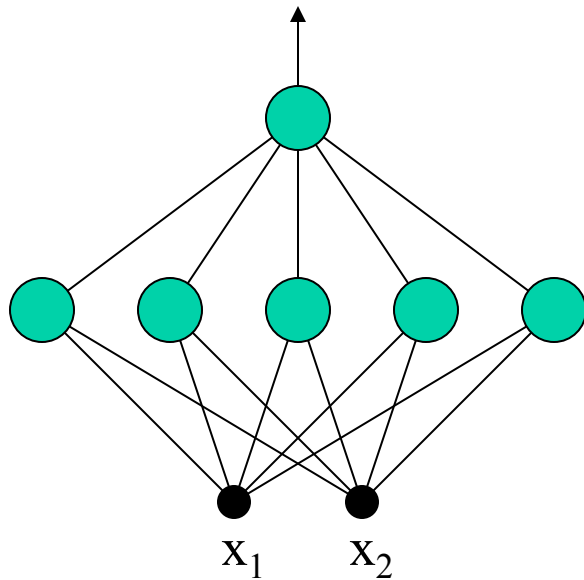
Reti a tre strati

- Sono in gradi di separare regioni convesse numero di lati \leq numero neuroni nascosti



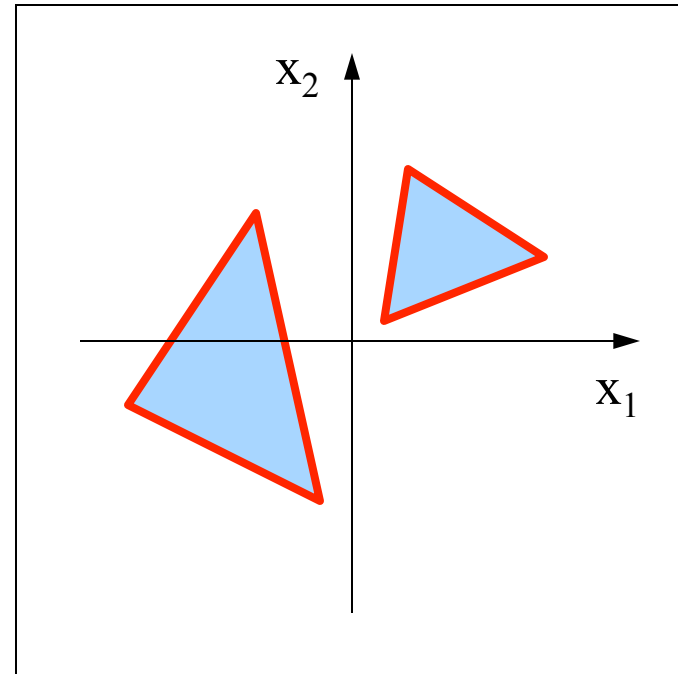
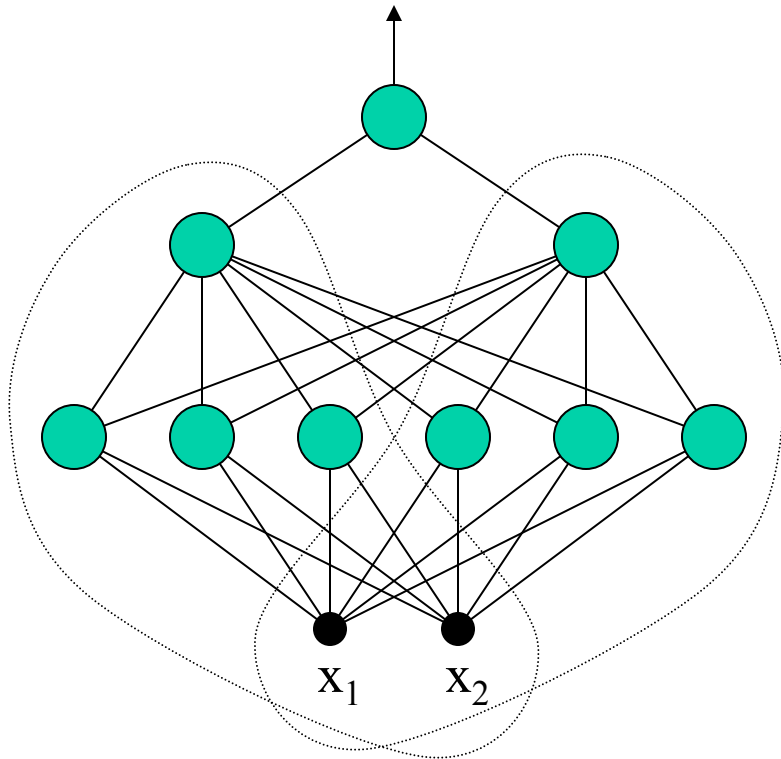
Reti a tre strati

- Sono in gradi di separare regioni convesse numero di lati \leq numero neuroni nascosti



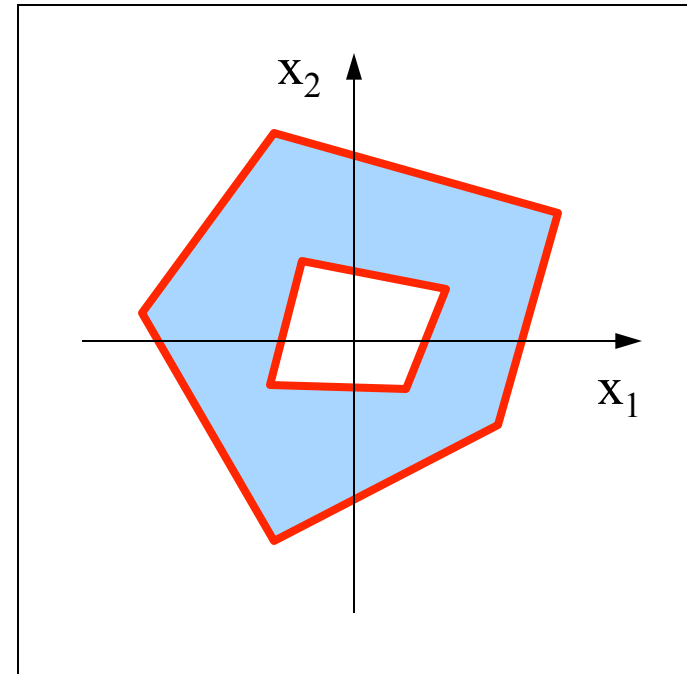
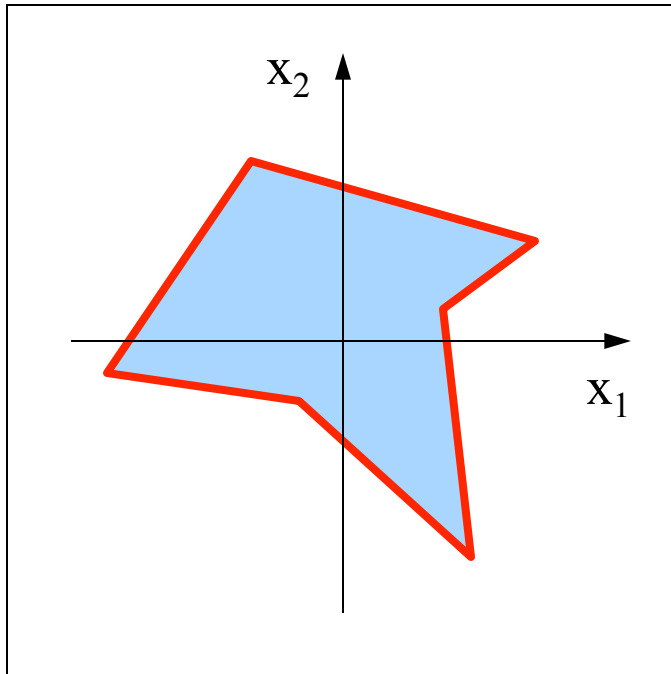
Reti a quattro strati

- Sono in gradi di separare regioni qualsiasi:



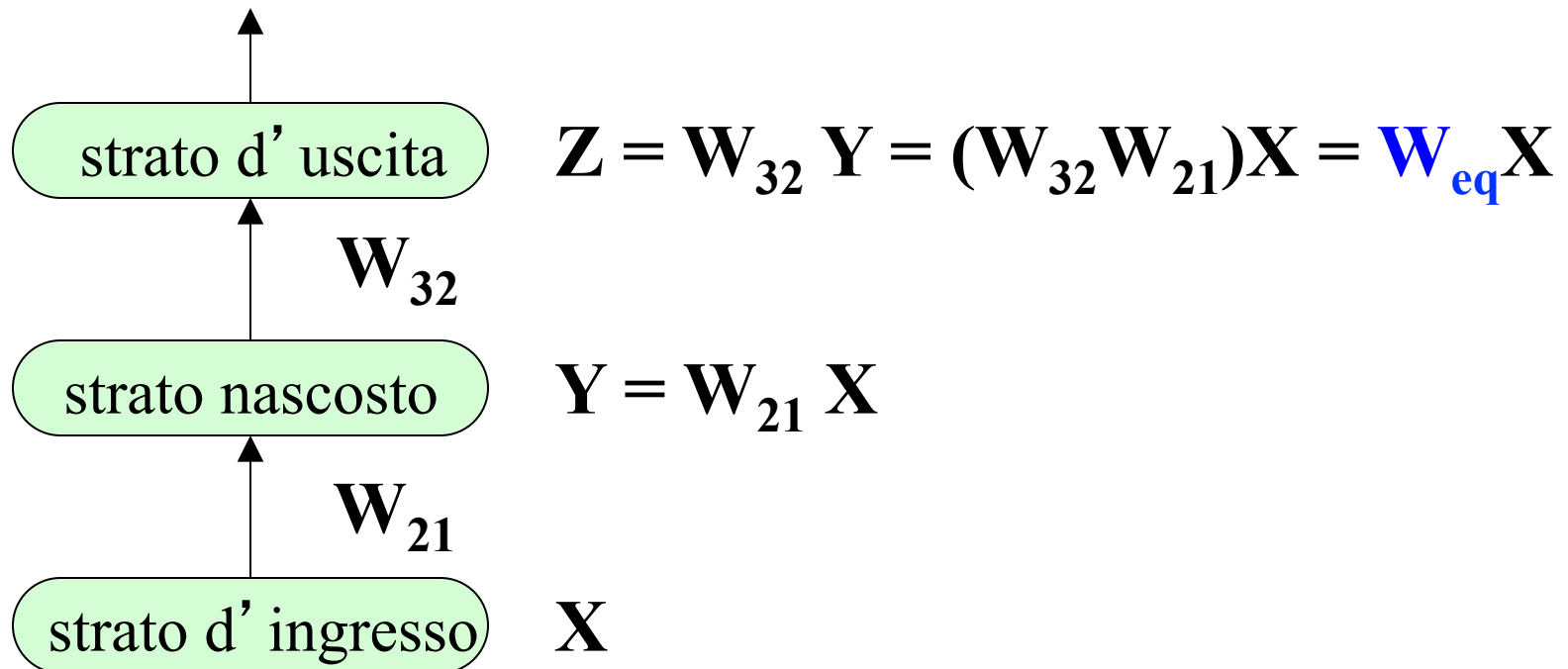
Reti a quattro strati

- L'aggiunta di altri strati non migliora la capacità di classificazione.



Importanza della non linearità

- Se le funzioni di uscita fossero lineari, una rete a N strati sarebbe sempre riconducibile a 2 strati:



Implicazioni

Per effettuare classificazioni complesse, i neuroni devono essere **non lineari** ed essere organizzati su **più strati**.

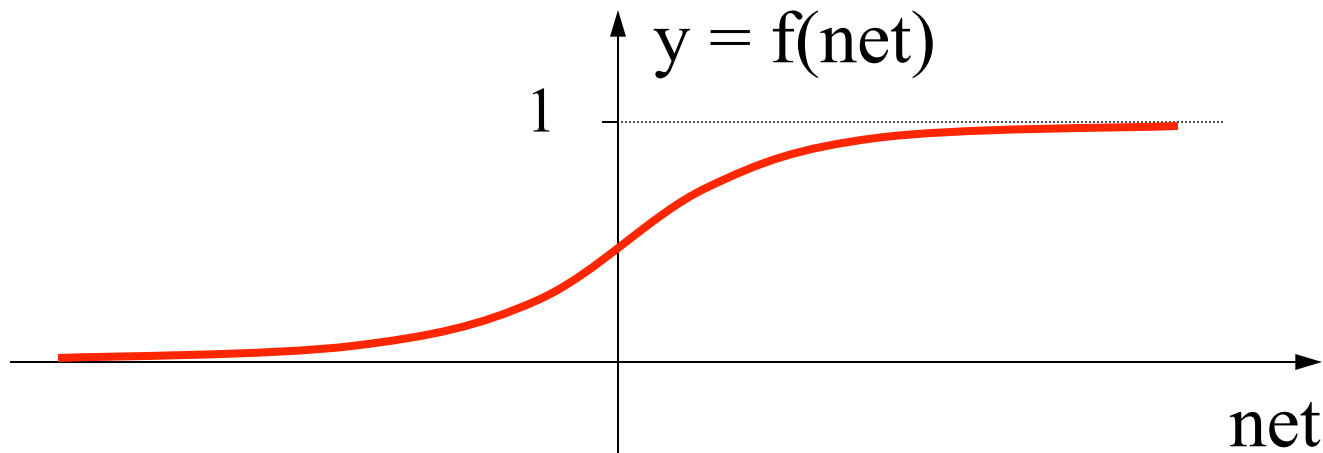
Problemi

- Come si addestra una rete multistrato?
- Qual è l'uscita desiderata dei neuroni nascosti?

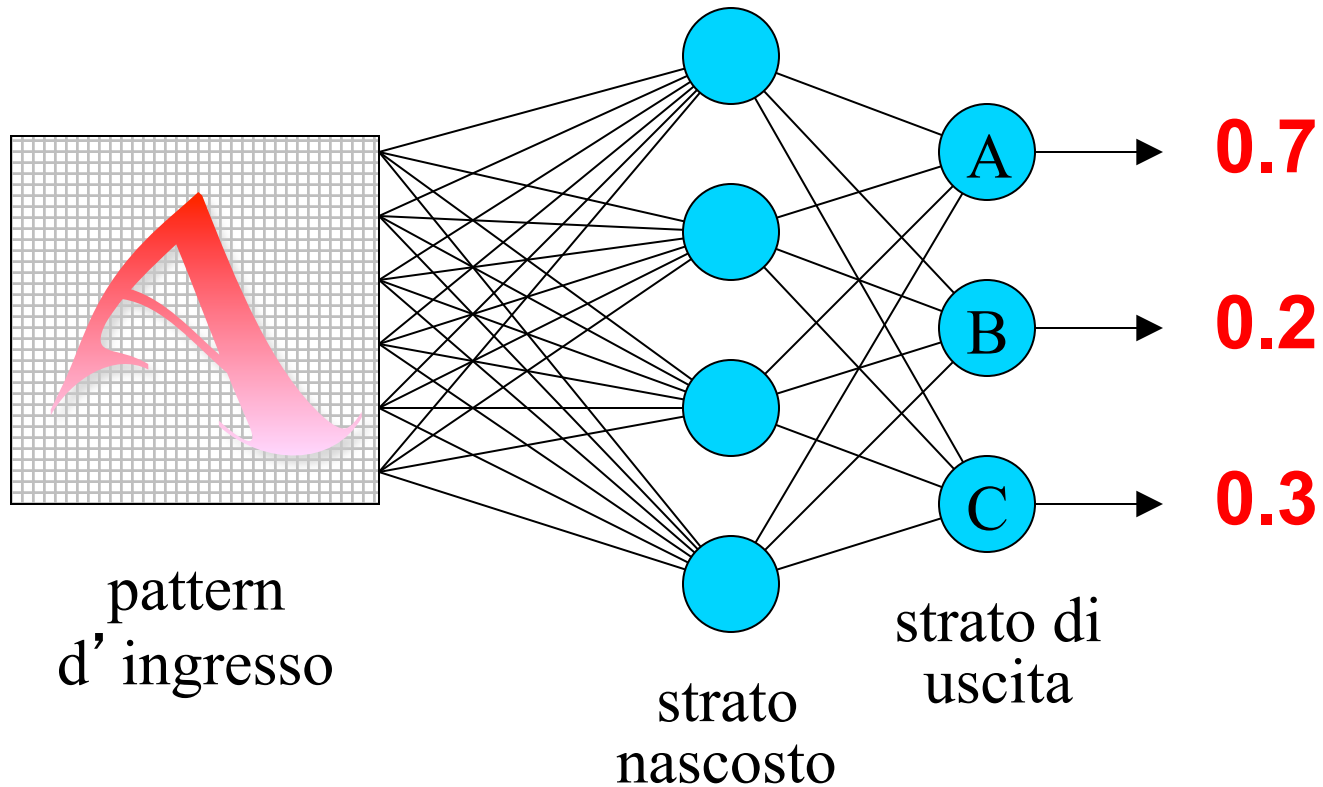
Back Propagation

(Rumelhart-Hinton-Williams, '85)

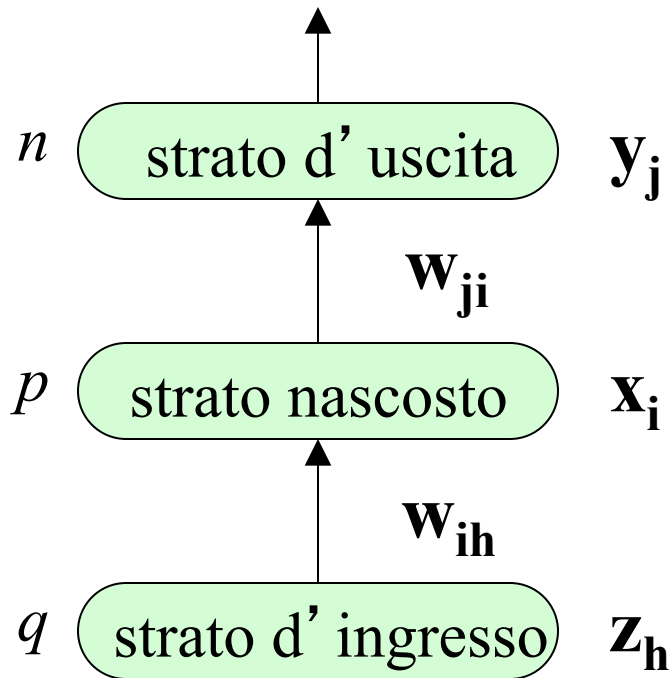
- Reti stratificate
- Ingressi a valori reali $\in [0,1]$
- Neuroni non lineari con funzione di uscita sigmoideale (deve essere derivabile):



Riconoscitore di caratteri



Back Propagation: Definizioni



t_j uscita desiderata

Errore sull' esempio k

$$E_k = \sum_{j=1}^n (t_{kj} - y_{kj})^2$$

Errore globale

$$E = \sum_{k=1}^M E_k$$

Training Set

$$TS = \{(\mathbf{X}_k, \mathbf{y}_{dk}), \quad k = 1, M\}$$

Back Propagation: Obiettivi

- **Apprendimento**

insegnare alla rete un insieme di associazioni desiderate (X_k, t_k) : **Training Set (TS)**

- **Convergenza**

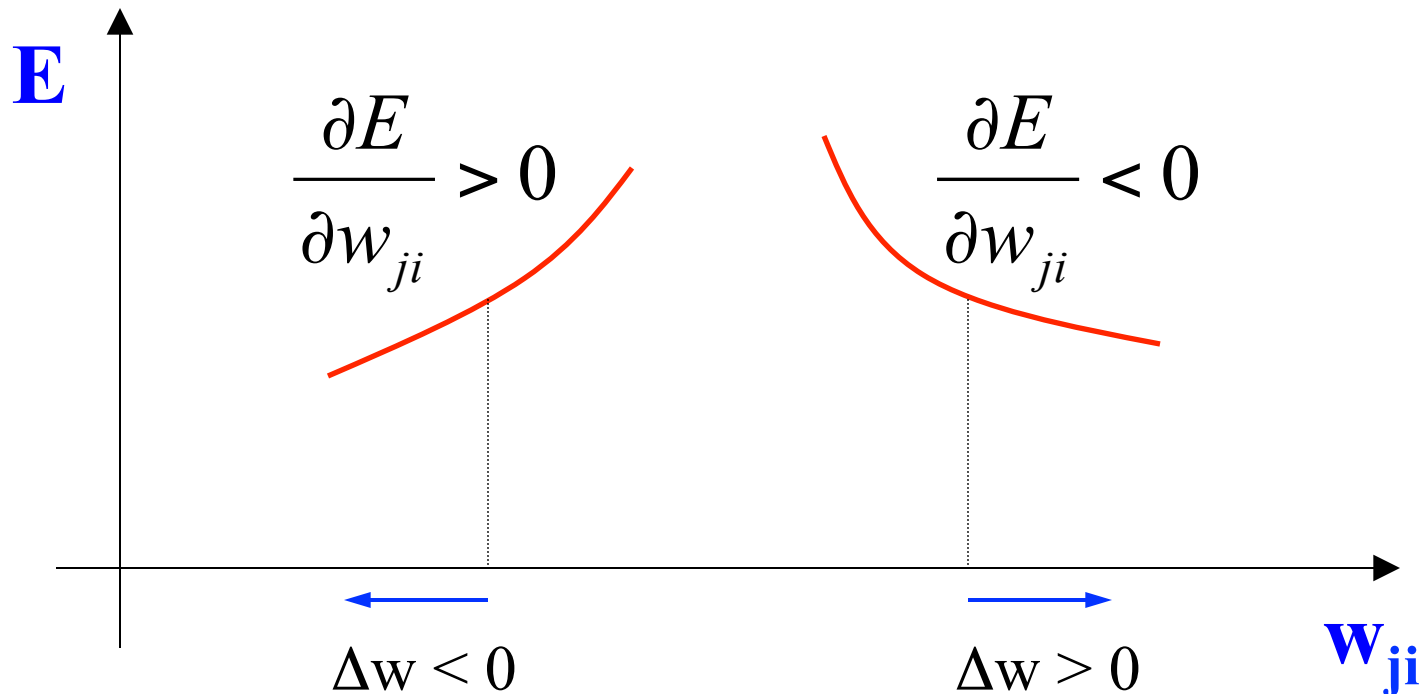
ridurre l'errore globale **E** al variare dei pesi, in modo che **E** < ϵ

- **Generalizzazione**

far si che la rete si comporti bene su esempi mai visti.

Convergenza

Per ridurre l'errore al variare dei pesi, si adotta il metodo di discesa del gradiente:



Aggiornamento dei pesi

Dunque i pesi sono variati in base alla seguente legge:

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

Regola del
gradiente

η = coefficiente di apprendimento (learning rate)

Sviluppo con l'errore E_k

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

essendo $net_j = \sum_{i=1}^p w_{ji} x_i$ si ha:

$$\frac{\partial net_j}{\partial w_{ji}} = x_i$$

e posto $\delta_j = -\frac{\partial E_k}{\partial net_j}$ si ha:

$$\Delta w_{ji} = \eta \delta_j x_i$$

Calcolo di δ_j (neuroni di uscita)

$$\delta_j = -\frac{\partial E_k}{\partial net_j} = -\frac{\partial E_k}{\partial y_j} \frac{\partial y_j}{\partial net_j}$$

essendo $E_k = \sum_{j=1}^n (t_{kj} - y_{kj})^2$ si ha:

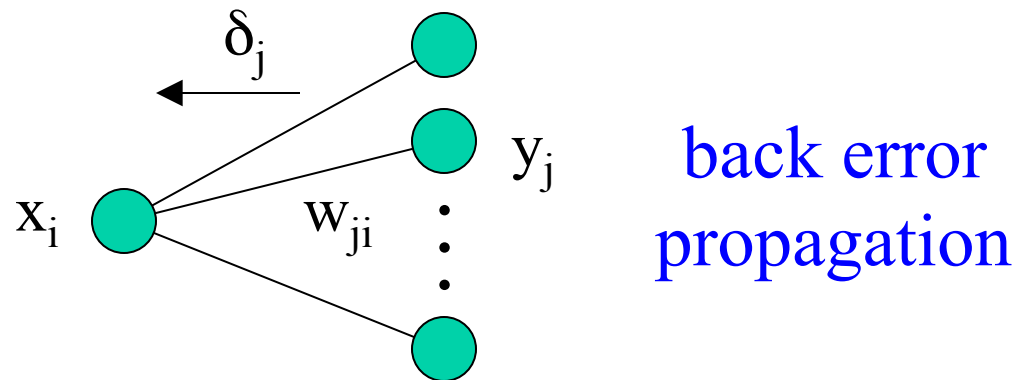
$$\frac{\partial E_k}{\partial y_j} = -(t_{kj} - y_{kj})$$

e poiché $\frac{\partial y_j}{\partial net_j} = f'(net_j)$ si ottiene:

$$\delta_j = (t_{kj} - y_{kj}) f'(net_j)$$

Calcolo di δ_j (neuroni nascosti)

La formula $\delta_i = (x_{di} - x_i) f'(net_i)$
non si può usare perché x_{di} non è noto.



$$\delta_i = f'(net_i) \sum_{j=1}^n \delta_j w_{ji}$$

Aggiornamento dei pesi

$$\Delta w_{ji} = \eta \delta_j x_i$$

Per lo strato
di uscita:

$$\delta_j = (t_j - y_j) f'(net_j)$$

Per lo strato
nascosto

$$\delta_i = f'(net_i) \sum_{j=1}^n \delta_j w_{ji}$$

Back Propagation: Algoritmo

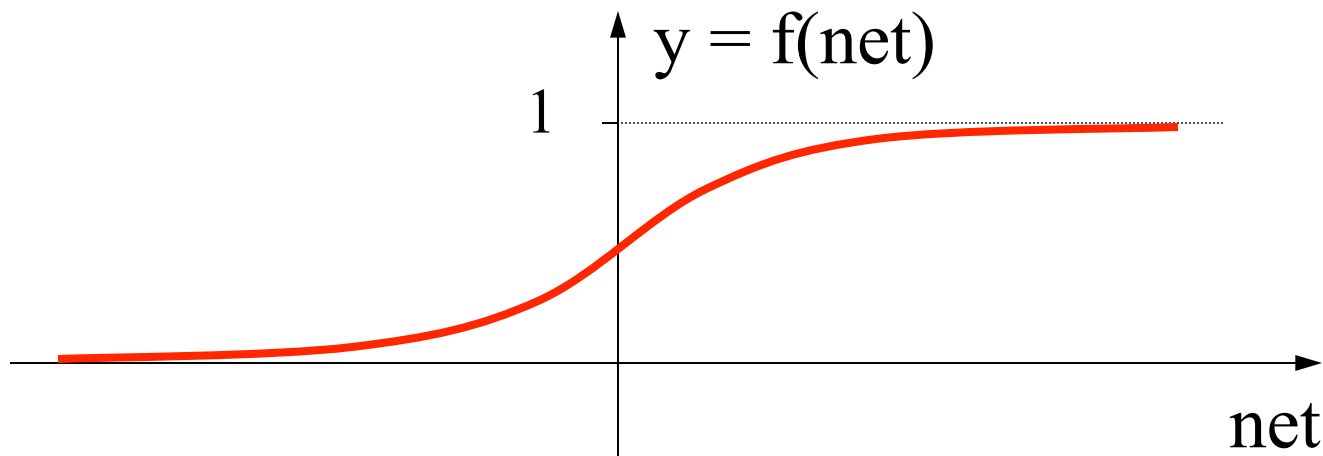
1. inizializza i pesi in modo casuale;
2. **do** {
3. inizializza l'errore globale $\mathbf{E} = \mathbf{0}$;
4. **for each** $(X_k, t_k) \in \text{TS}$ {
5. calcola \mathbf{y}_k e l'errore \mathbf{E}_k ;
6. calcola i δ_j sullo strato di uscita;
7. calcola i δ_i sullo strato nascosto;
8. aggiorna i pesi della rete: $\Delta \mathbf{w} = \eta \delta \mathbf{x}$;
9. aggiorna l'errore globale: $\mathbf{E} = \mathbf{E} + \mathbf{E}_k$; }
10. } **while** $(\mathbf{E} > \epsilon)$;

Back Propagation: Osservazioni

- Per favorire l'apprendimento, i pesi devono essere inizializzati con valori piccoli. Infatti:

net piccola \Rightarrow **f'(net)** grande \Rightarrow **Δw** grande

$$\Delta w = \eta \delta x \propto f'(net)$$

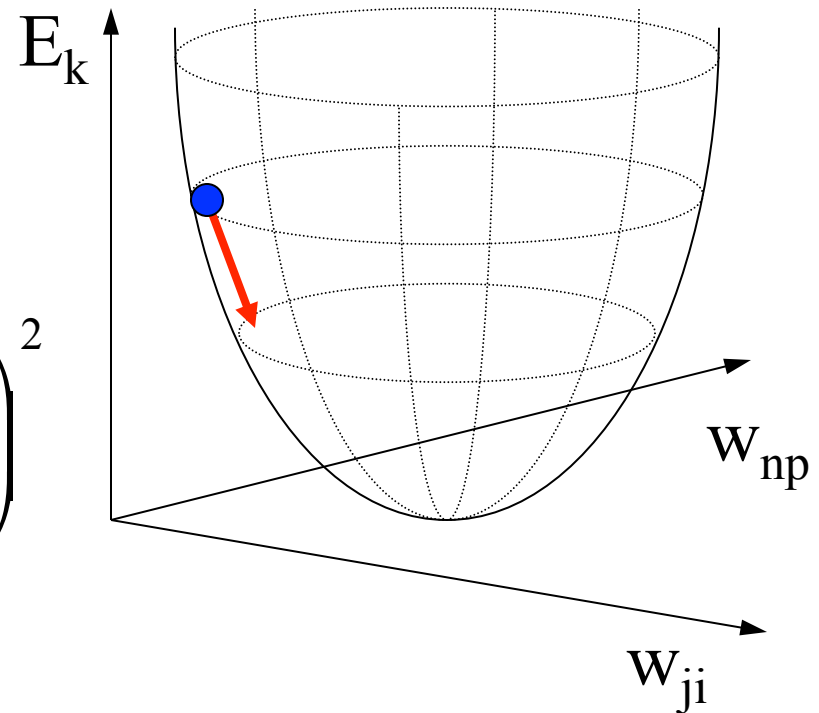


Back Propagation: Osservazioni

- L' errore ha una forma quadratica nello spazio dei pesi:

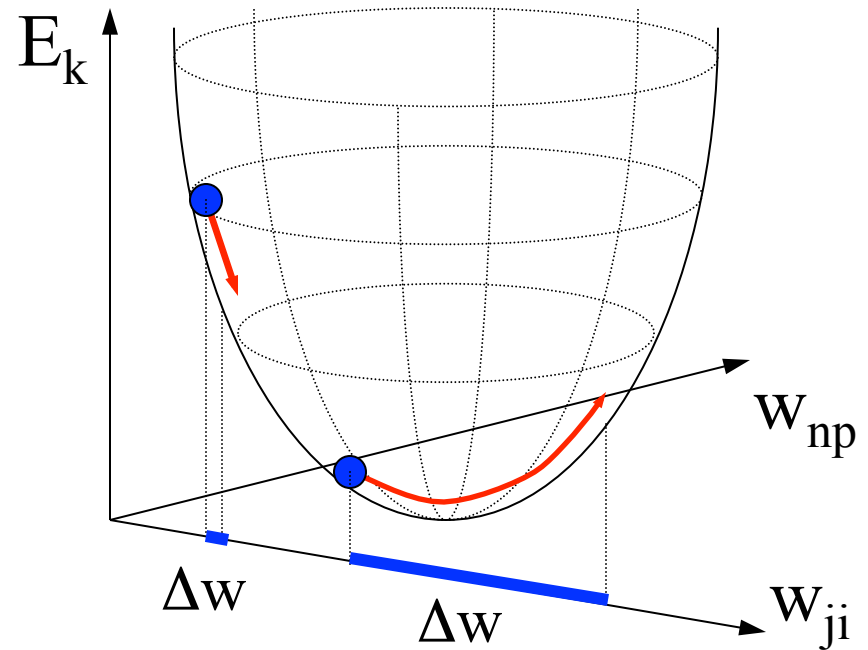
$$E_k = \sum_{j=1}^n (t_{kj} - y_{kj})^2$$

$$= \sum_{j=1}^n \left(t_{kj} - f \left(\sum_{i=1}^p w_{ji} x_{ki} \right) \right)^2$$



Back Propagation: Osservazioni

$$\Delta w_{ji} = \eta \delta_j x_i$$



η troppo piccolo \Rightarrow apprendimento lento

η troppo grande \Rightarrow oscillazioni

Possibili soluzioni

- Variare η in funzione dell'errore, in modo da accelerare la convergenza all'inizio e ridurre le oscillazioni alla fine.
- Smorzare le oscillazioni con un filtro passa basso sui pesi:

$$\Delta w_{ji}(t) = \eta \delta_j x_i + \alpha \Delta w_{ji}(t-1)$$

α è detto **momentum**

Back Propagation: Osservazioni

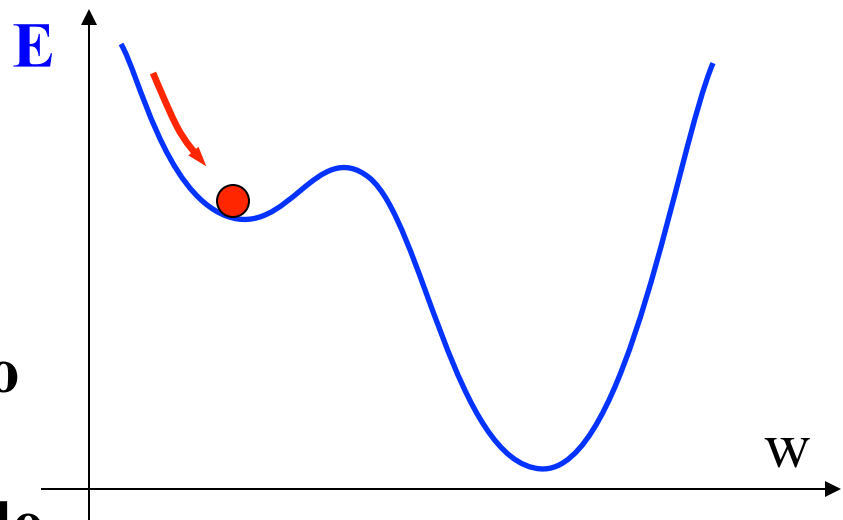
- La forma quadratica è distorta dalla funzione di uscita non lineare:

$$E_k = \sum_{j=1}^n (t_{kj} - y_{kj})^2 = \sum_{j=1}^n \left(t_{kj} - f\left(\sum_{i=1}^p w_{ji} x_{ki}\right) \right)^2$$

**Rischio di fermarsi
in un minimo locale**

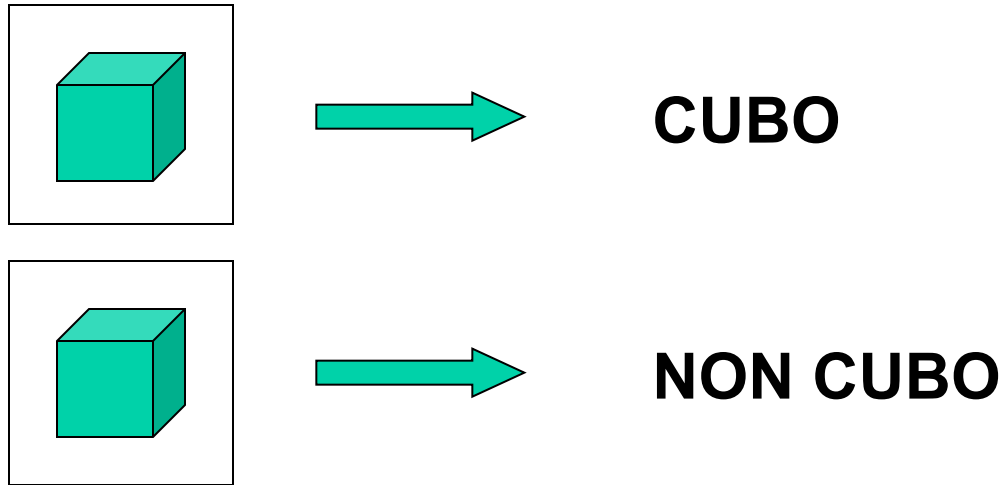


**Occorre ricominciare da capo
con nuovi pesi oppure
modificare alcuni pesi in modo
casuale**



Back Propagation: Osservazioni

- Se alcuni esempi sono inconsistenti, la convergenza dell'apprendimento non è garantita:



Nei casi reali, l'inconsistenza può essere introdotta dal rumore sui dati di ingresso.

Back Propagation: Osservazioni

- Gli esempi del TS non sono altro che campioni di una funzione ingresso/uscita $F:\mathcal{X}^q \rightarrow \mathcal{X}^n$ che descrive il problema associativo.
- Durante l'apprendimento tale funzione viene approssimata mediante una combinazione non lineare di sigmoidi.
- Il processo di approssimazione della rete è simile a quello della trasformata di Fourier.

Generalizzazione

- E' la capacità della rete di riconoscere stimoli leggermente diversi da quelli con cui è stata addestrata.
- Per valutare la capacità della rete di generalizzare gli esempi del TS, si definisce un altro insieme di esempi, detto **Validation Set** (VS).
- Terminato l'apprendimento sul TS ($E_{TS} < \epsilon$), si valuta l'errore sul VS (E_{VS}).

Generalizzazione

- Il numero di parametri da regolare dipende dal numero di neuroni nascosti della rete.
- Pochi neuroni nascosti potrebbero non essere sufficienti a ridurre l'errore globale.
- Troppi neuroni nascosti potrebbero fossilizzare eccessivamente la rete sugli esempi specifici del TS.
- La rete risponderebbe bene sul TS, ma l'errore sarebbe elevato su altri esempi (**overtraining**).

RETI DI KOHONEN

**Competitive Learning
&
Self Organizing Maps**

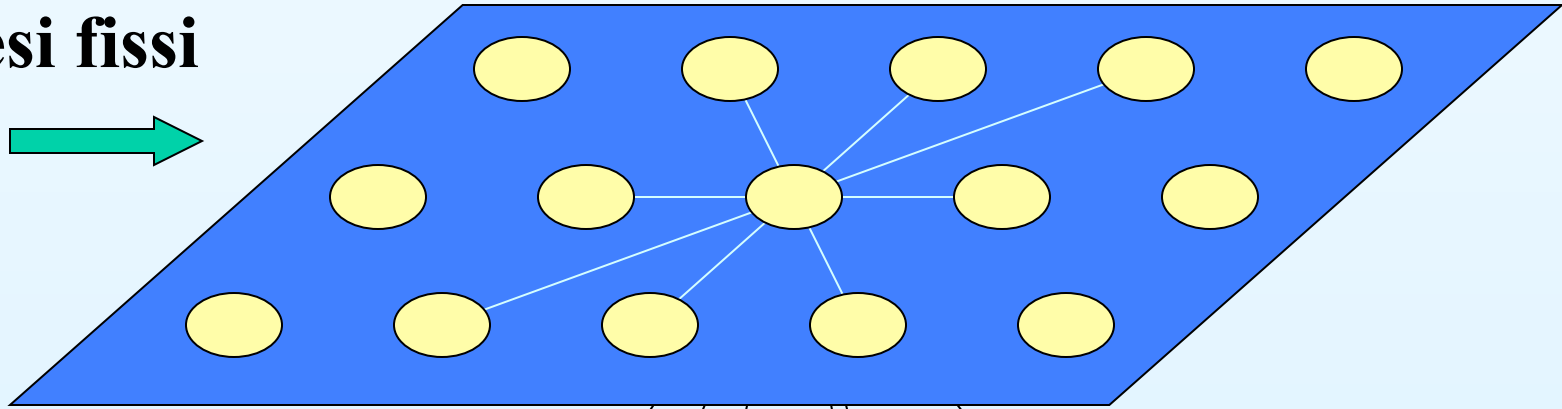
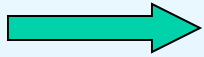
Reti di Kohonen

Nel 1983, Teuvo Kohonen riuscì a costruire un modello neurale in grado di replicare il processo di formazione delle mappe sensoriali sulla corteccia cerebrale:

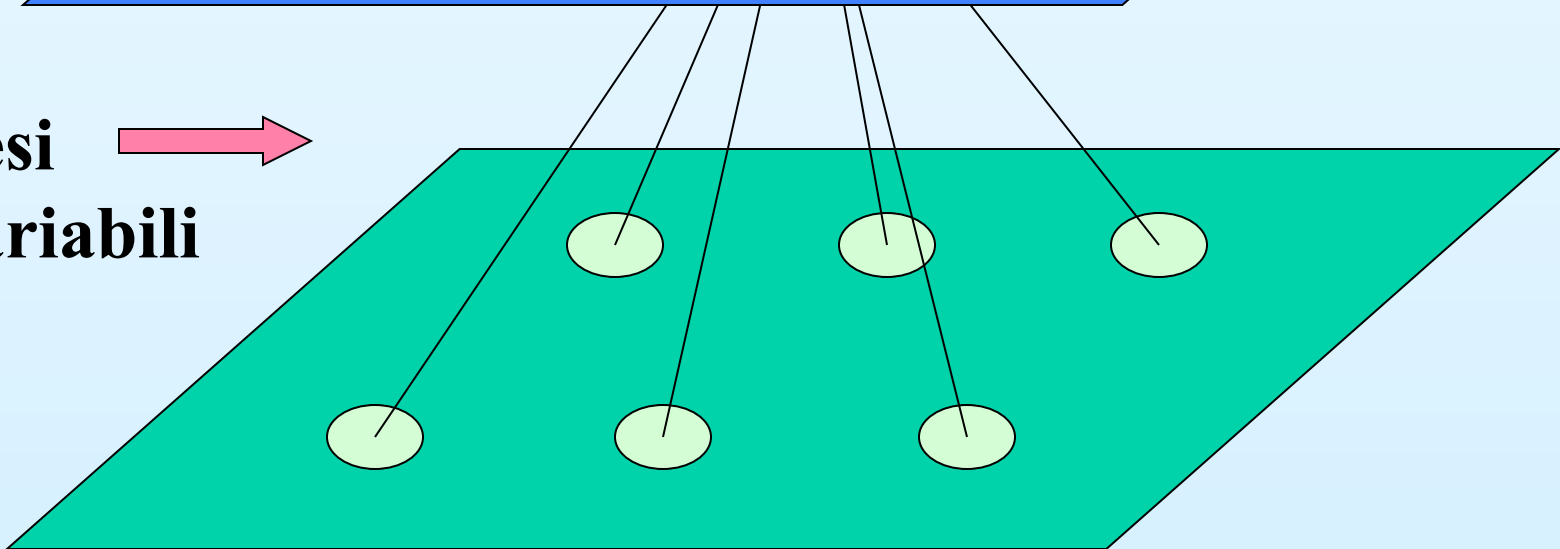
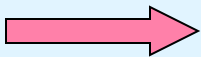
- rete stratificata
- apprendimento senza supervisione basato sulla competizione fra neuroni

Architettura

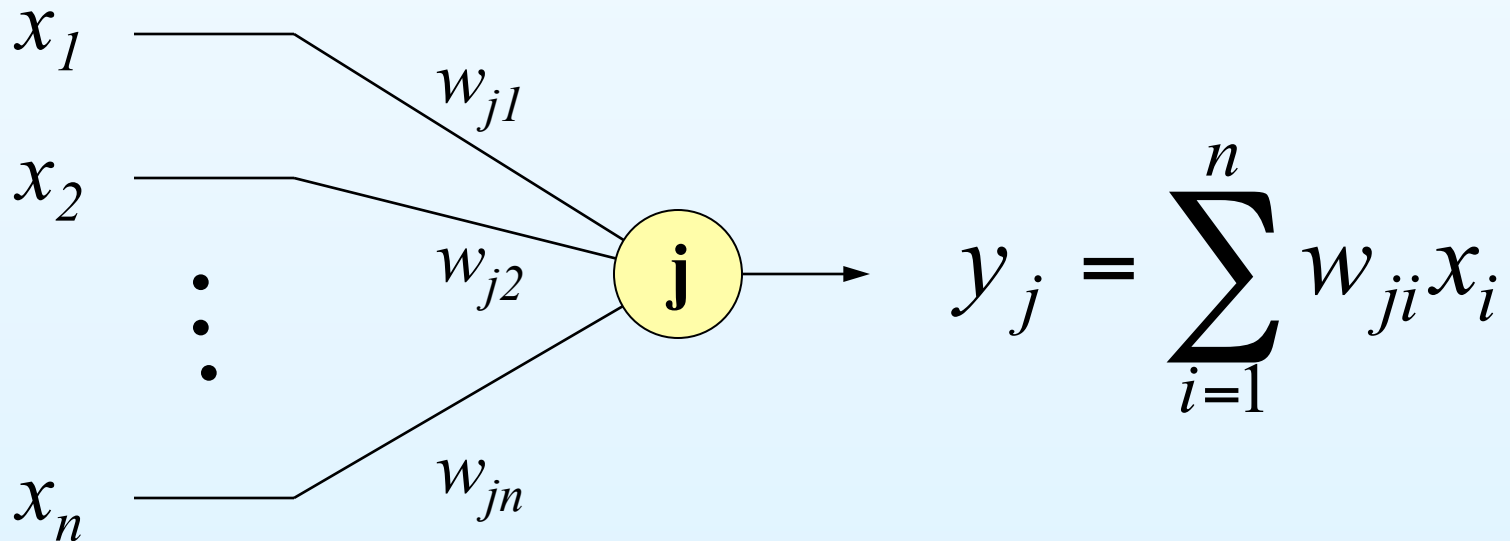
pesi fissi



**pesi
variabili**



Neuroni lineari

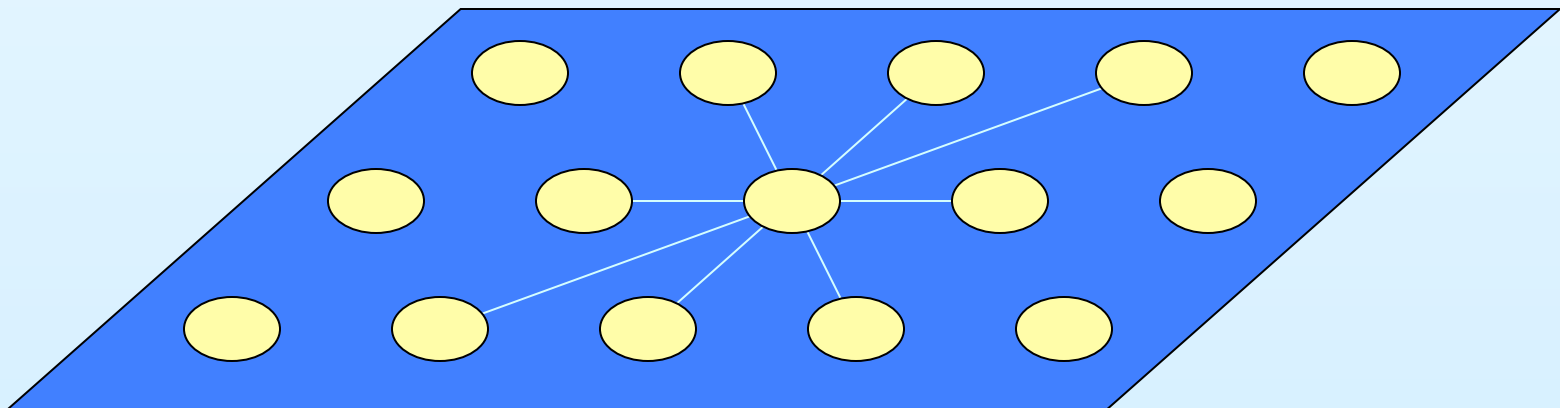


$$y_j = \sum_{i=1}^n w_{ji} x_i = W_j \cdot X = |W_j| \cdot |X| \cos \theta$$

Distribuzione dei pesi fissi

I pesi fissi dipendono dalla distanza dei neuroni:

- **neuroni vicini \Rightarrow pesi positivi**
- **neuroni lontani \Rightarrow pesi negativi**



Apprendimento competitivo

- Grazie all'inibizione laterale, i neuroni competono per rispondere ad uno stimolo.
- Il neurone con uscita maggiore vince la competizione e si specializza a riconoscere lo stimolo.
- Grazie alle connessioni eccitatorie, i neuroni vicini al vincitore risultano sensibili ad ingressi simili.

Si crea un isomorfismo tra
spazio di ingresso e spazio di uscita

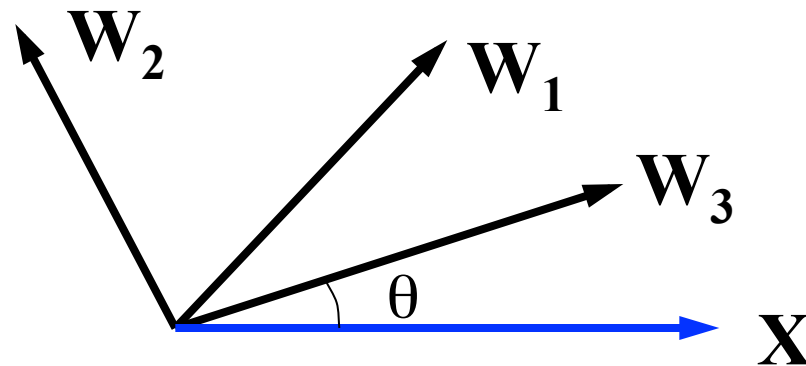
Implementazione

- In realtà, per problemi di efficienza, i neuroni di uscita non vengono connessi tra loro.
- Il neurone vincitore viene scelto con una strategia globale confrontando le uscite di tutti i neuroni.
- Si possono usare due tecniche:
 1. Si sceglie il neurone con uscita massima;
 2. Si sceglie il neurone i cui vettore dei pesi è più simile all'ingresso presentato.

Neurone vincitore (metodo 1)

Il neurone vincente sull'ingresso X è quello che ha l'uscita maggiore:

$$y_j = \sum_{i=1}^n w_{ji} x_i = W_j \bullet X = |W_j| |X| \cos \theta$$



Definizione di distanza

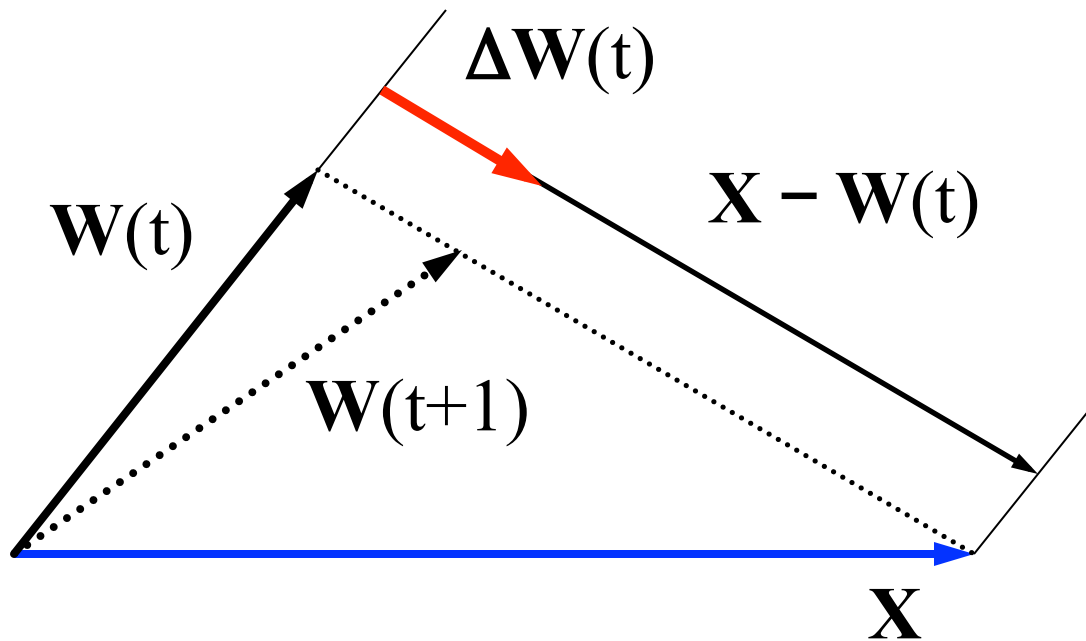
Distanza Euclidea: $DIS(X, Y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$

**Distanza stradale:
(o di Manhattan)** $DIS(X, Y) = \sum_{i=1}^n |x_i - y_i|$

Distanza di Hamming: $DIS(X, Y) = \sum_{i=1}^n (x_i \neq y_i)$
(solo per vettori binari)

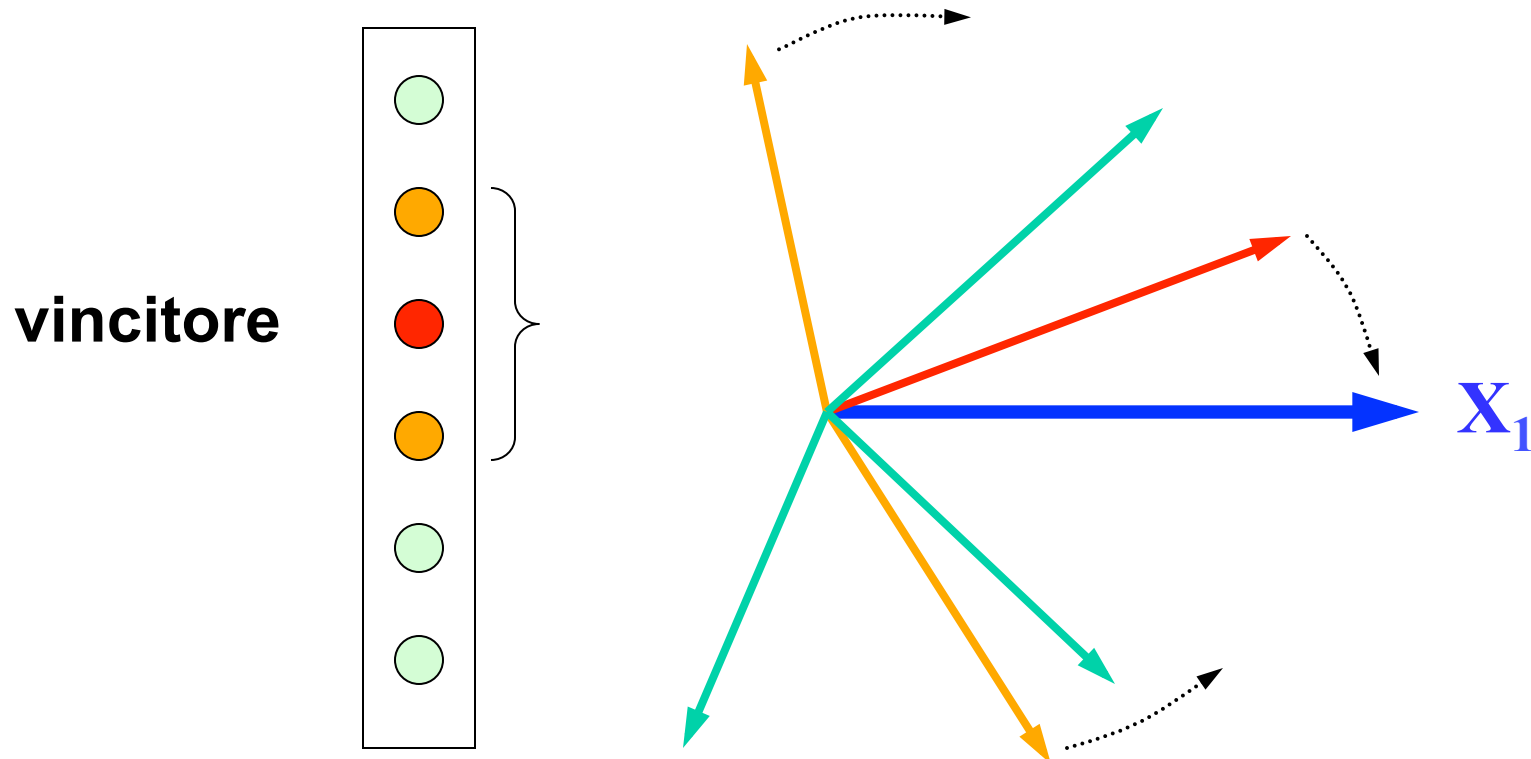
Legge di apprendimento

$$\Delta \mathbf{W}(t) = \alpha (\mathbf{X} - \mathbf{W})$$



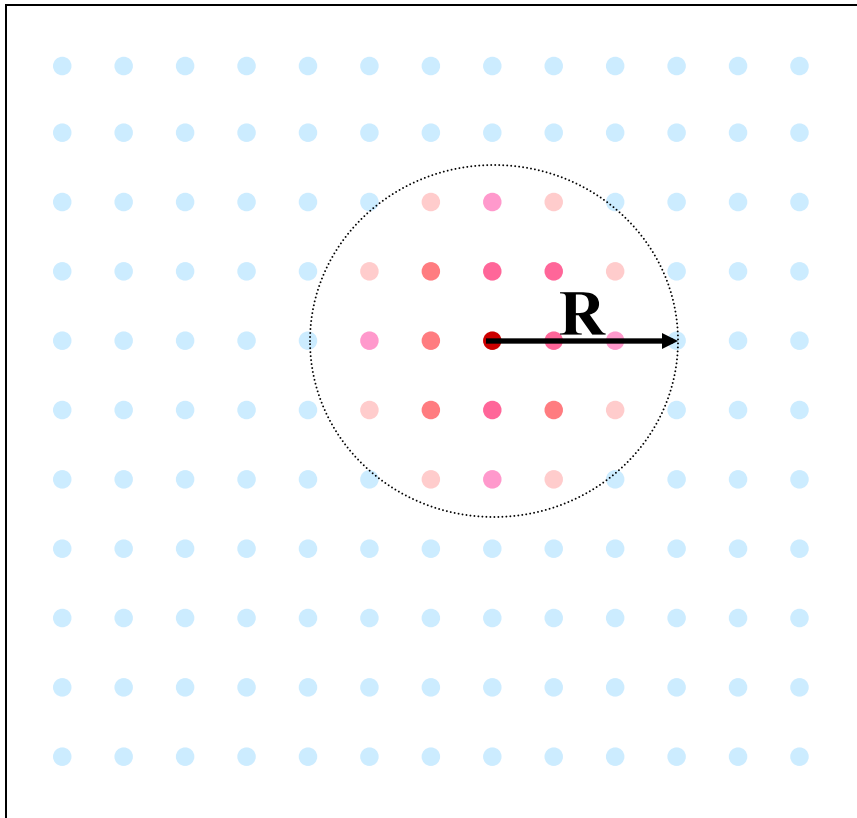
Aggiornamento del vicinato

Per simulare le connessioni radiali, si variano i pesi dei neuroni vicini al vincitore:



Raggio di interazione

Il vicinato è l'insieme di neuroni aventi una distanza dal vincitore minore di R :



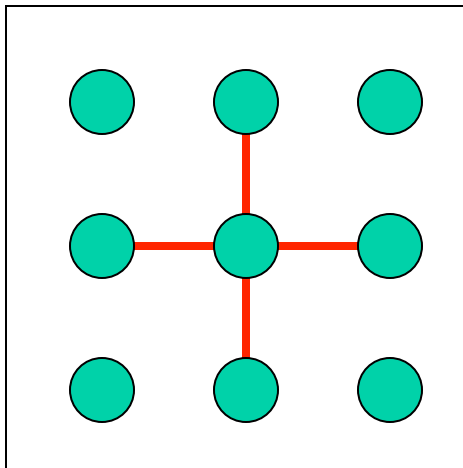
R = raggio di
interazione

Definizione della mappa

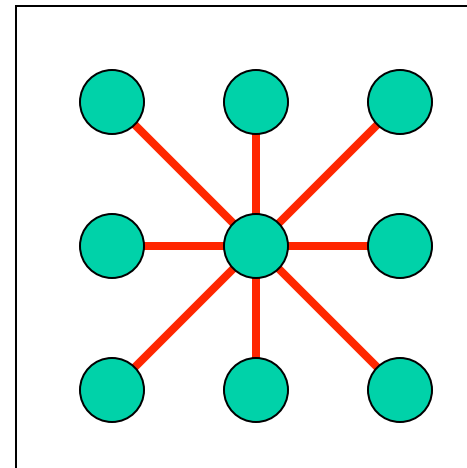
- Per consentire la formazione della mappa, occorre definire una topologia sullo strato di uscita.
- Ogni neurone deve avere una posizione identificata da un vettore di coordinate.
- La mappa di uscita è di solito definita come uno spazio a una o a due dimensioni.

Tipologie di vicinato

Vicinanza 4

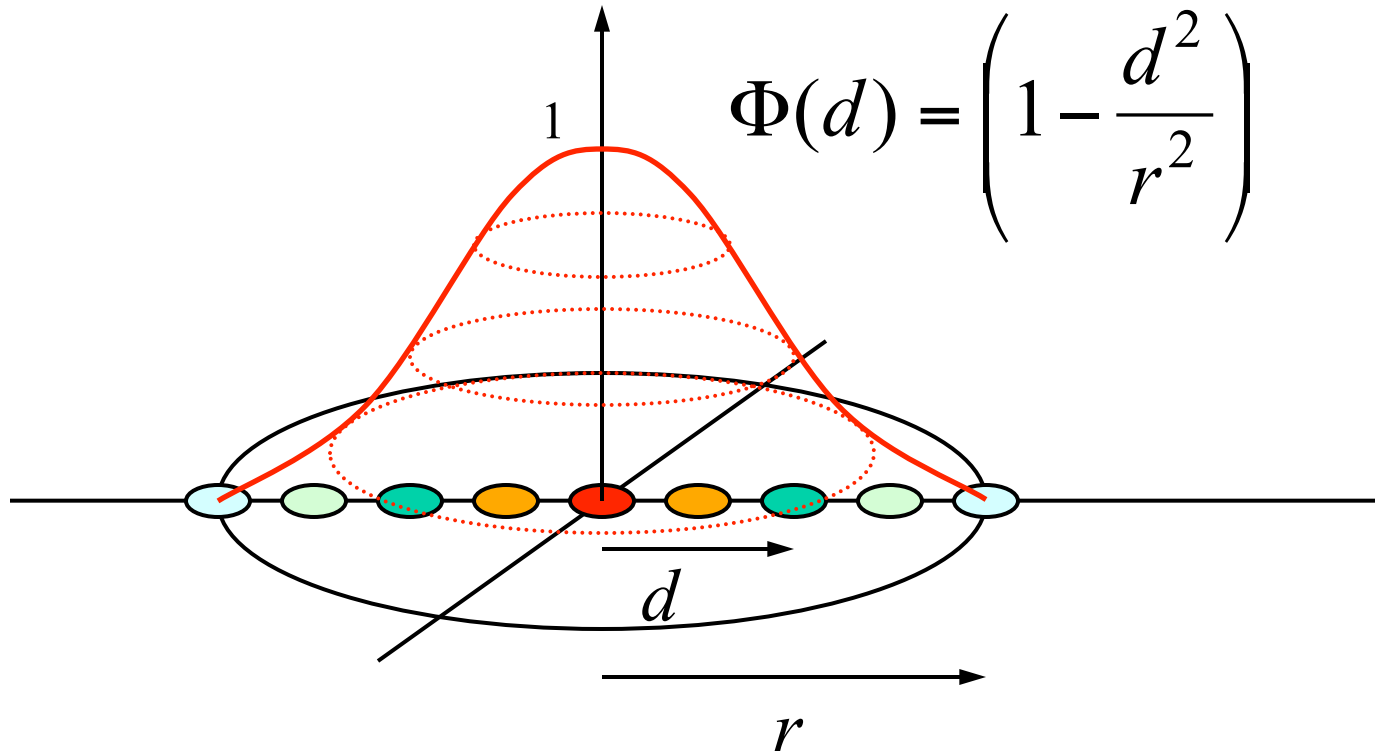


Vicinanza 8



Variazione dei pesi

I pesi dei neuroni vengono variati in base alla loro distanza dal neurone vincitore:



Variazione dei pesi

Detto j_0 l'indice del neurone vincitore, si ha:

$$\forall j \in \text{vicinato}(j_0, r)$$

$$d = \text{DIS}(j, j_0)$$

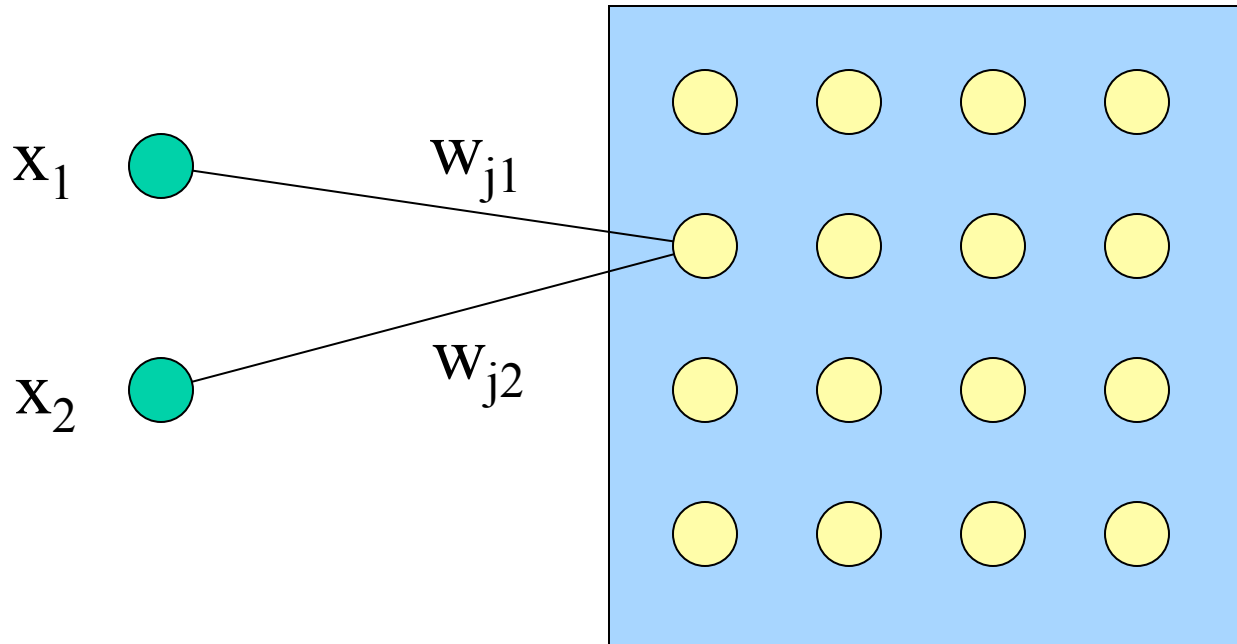
$$\Delta w_j = \alpha \Phi(d) (X - w_j)$$

Le quantità r e α variano (decrescono) nel tempo.

Algoritmo di apprendimento

1. Si inizializzano i pesi in modo casuale;
2. Si inizializzano i parametri: $\alpha = A$; $r = R$;
2. **do** {
4. **for each** ($X_k \in TS$) {
5. si calcolano tutte le uscite y_j ;
6. si determina il neurone vincitore j_0 ;
7. si aggiornano i pesi del vicinato;
8. }
9. si riducono α e r ;
10. } **while** ($\alpha > \alpha_{\min}$);

Esempio



Ingresso = vettore di coordinate su un piano

Mappa = Griglia con vicinanza 4

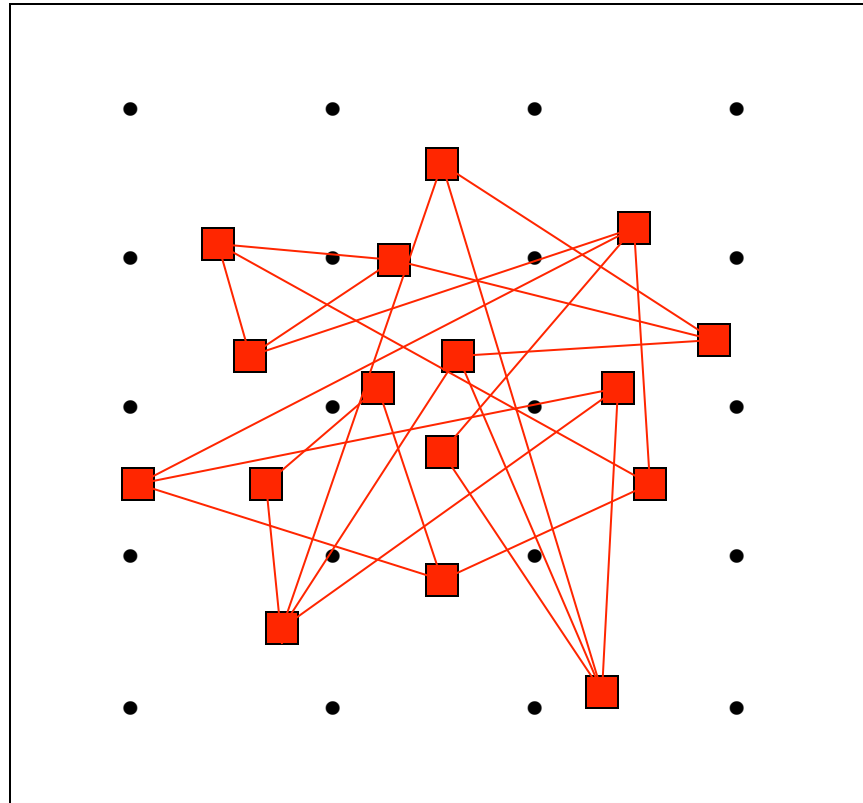
Stato Iniziale

$$N = M$$

- ingresso

■ peso

{ M ingressi
N neuroni



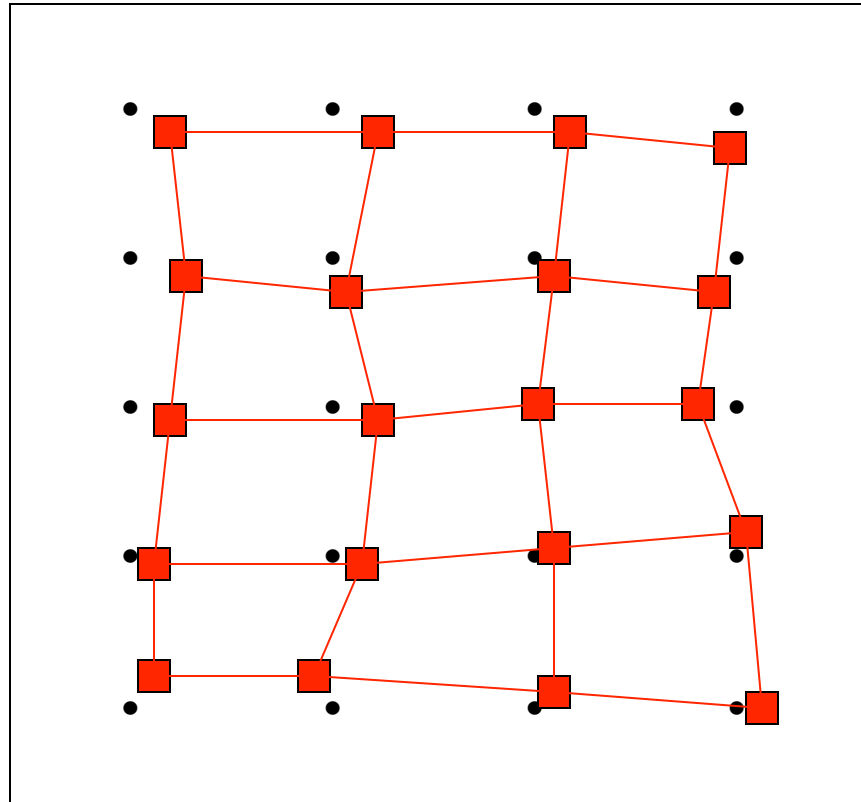
Stato finale

$$N = M$$

- ingresso

■ peso

{ M ingressi
N neuroni



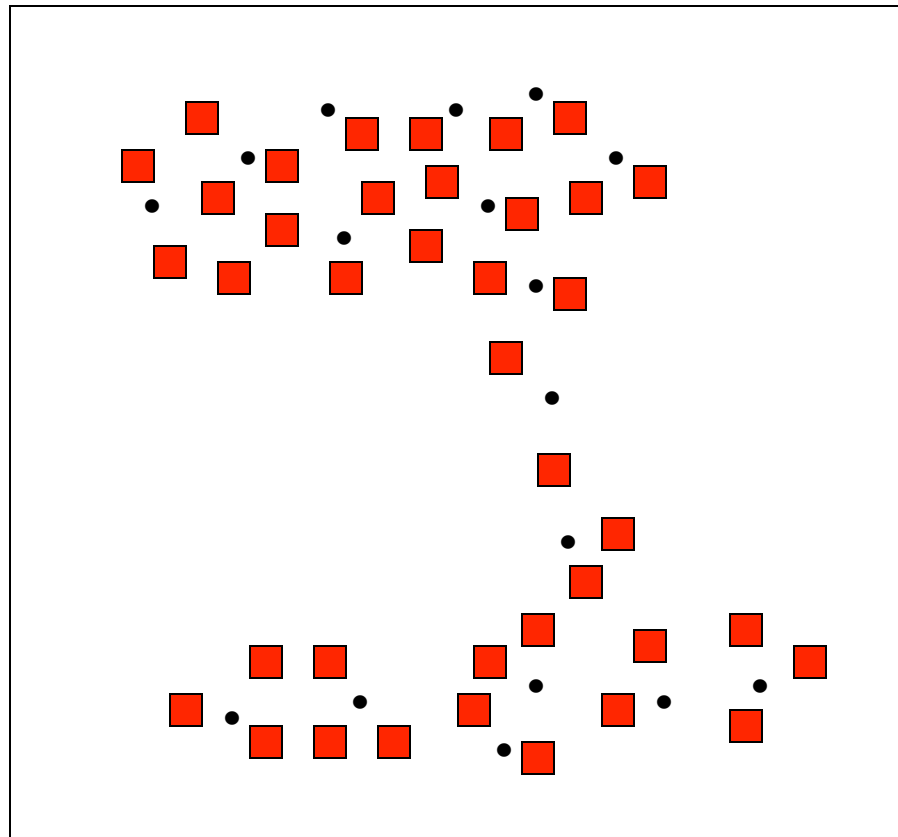
Esempio

$$N > M$$

- ingresso

■ peso

{ M ingressi
N neuroni



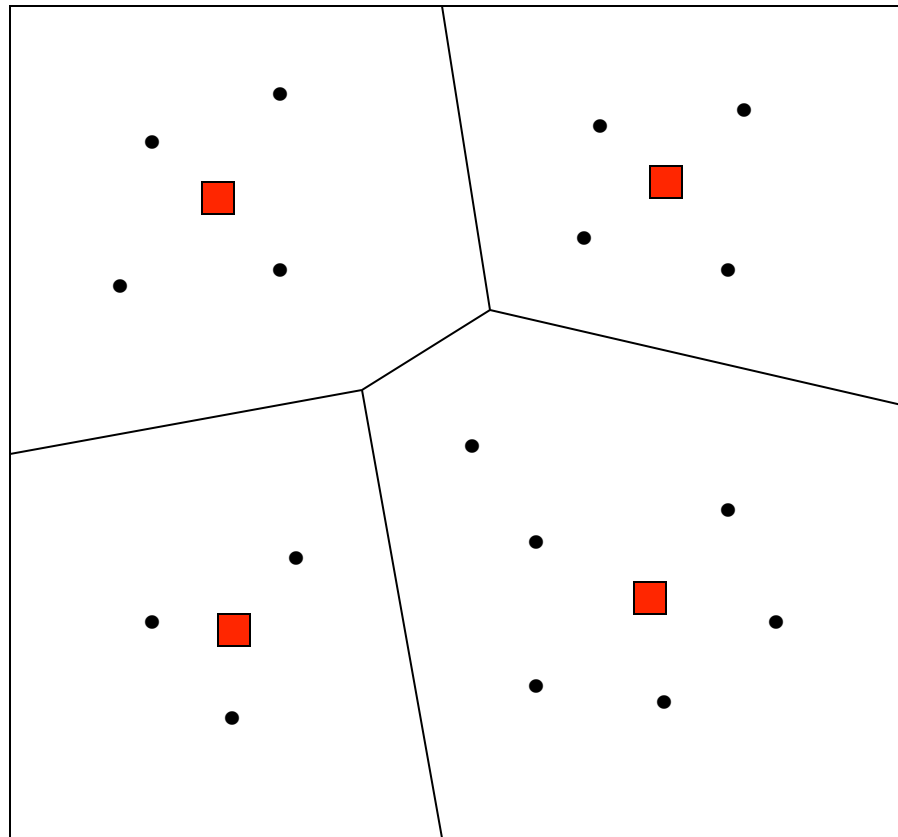
Esempio

$$N < M$$

- ingresso

■ peso

{ M ingressi
N neuroni

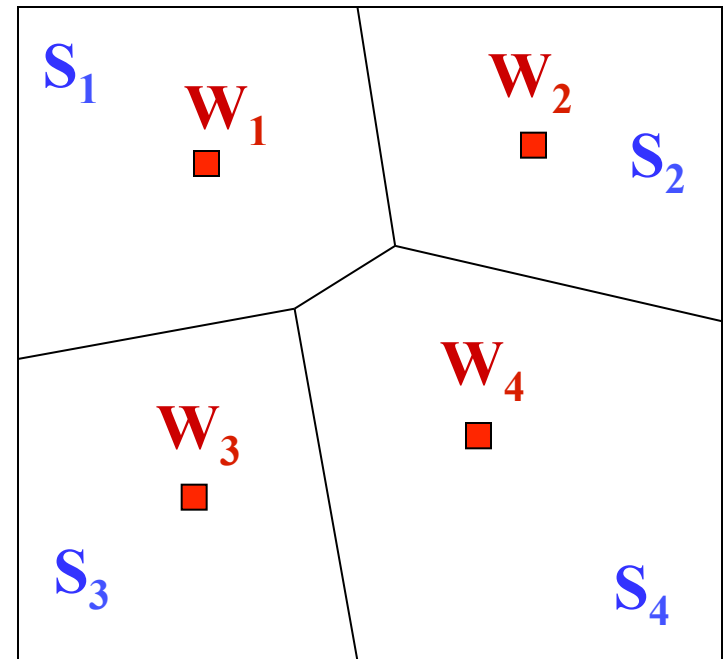


Tassellazione di Voronoi

E' una suddivisione di uno spazio S in n sottospazi S_i , tali che:

$W_i = \text{centroide di } S_i$

$$\left\{ \begin{array}{l} \bigcup_{i=1}^n S_i = S \\ S_i \cap S_j = \emptyset \quad \forall i \neq j \\ DIST(X_k, W_i) < DIST(X_k, W_j) \quad \forall i \neq j, X_k \in S_i \end{array} \right.$$



Applicazioni

Clustering

- Raggruppare un insieme enorme di dati in un numero limitato di classi, in base alla somiglianza dei dati.

Compression

- Convertire un'immagine con milioni di colori in un'immagine compressa su 256 livelli (non fissi).

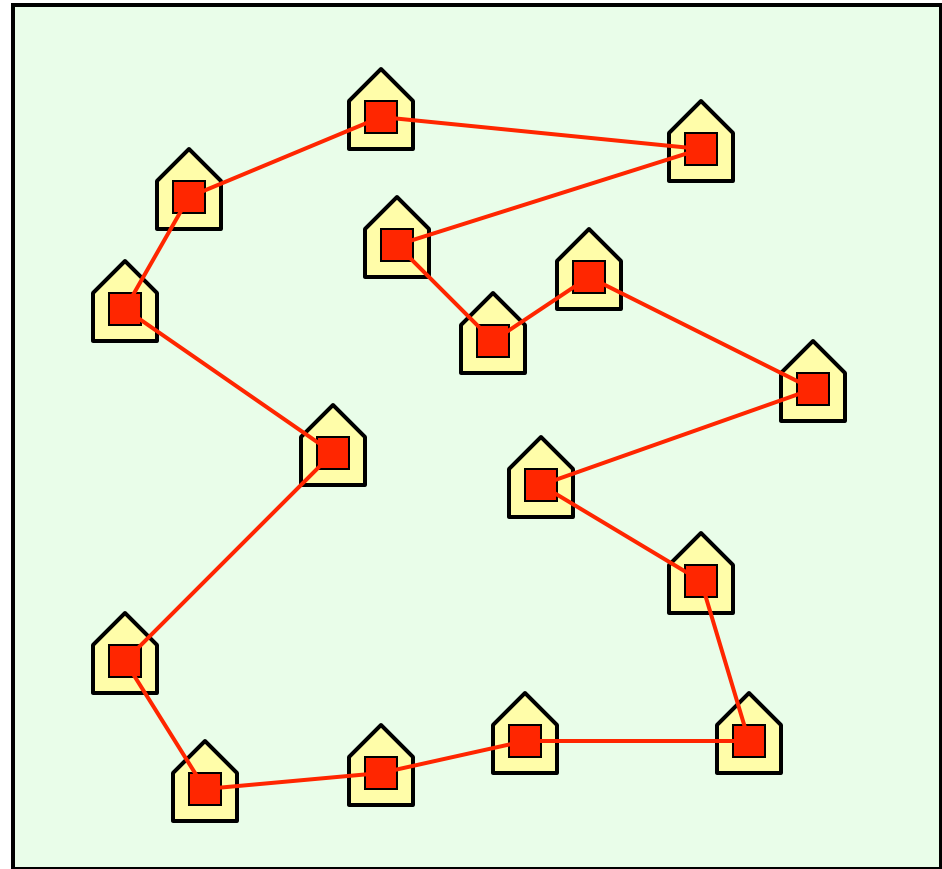
Classification

- Classificare un insieme di frasi in un insieme di classi distinte per argomenti correlati.
- Usato da alcuni motori di ricerca per classificare i gusti degli utenti collegati.

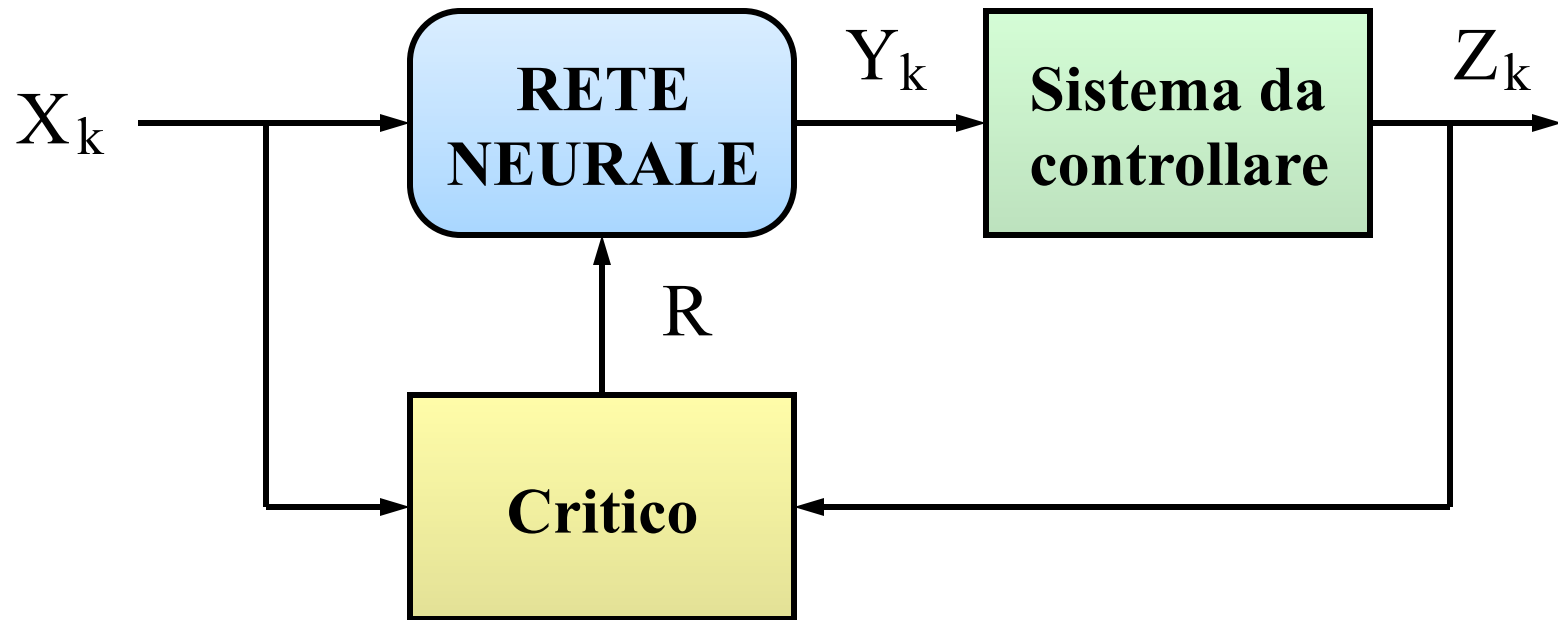
Ottimizzazione

Problema del commesso viaggiatore

{ Inputs: 2
Outputs: n
Map: 1D-mesh

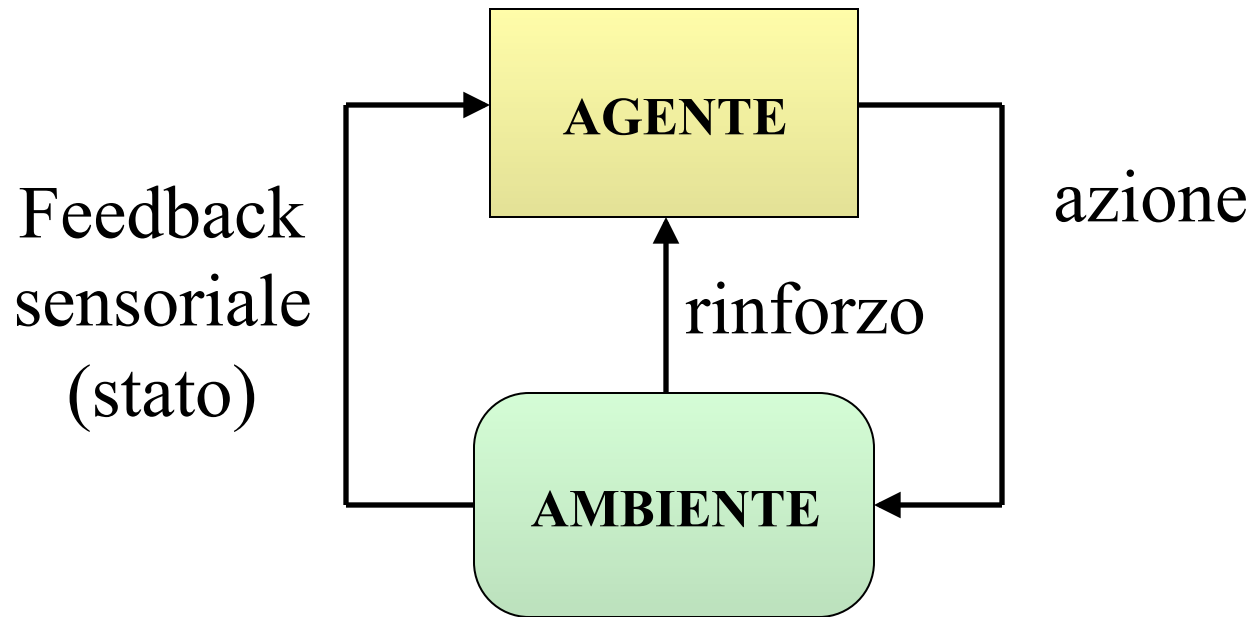


Apprendimento con rinforzo



Premi e punizioni

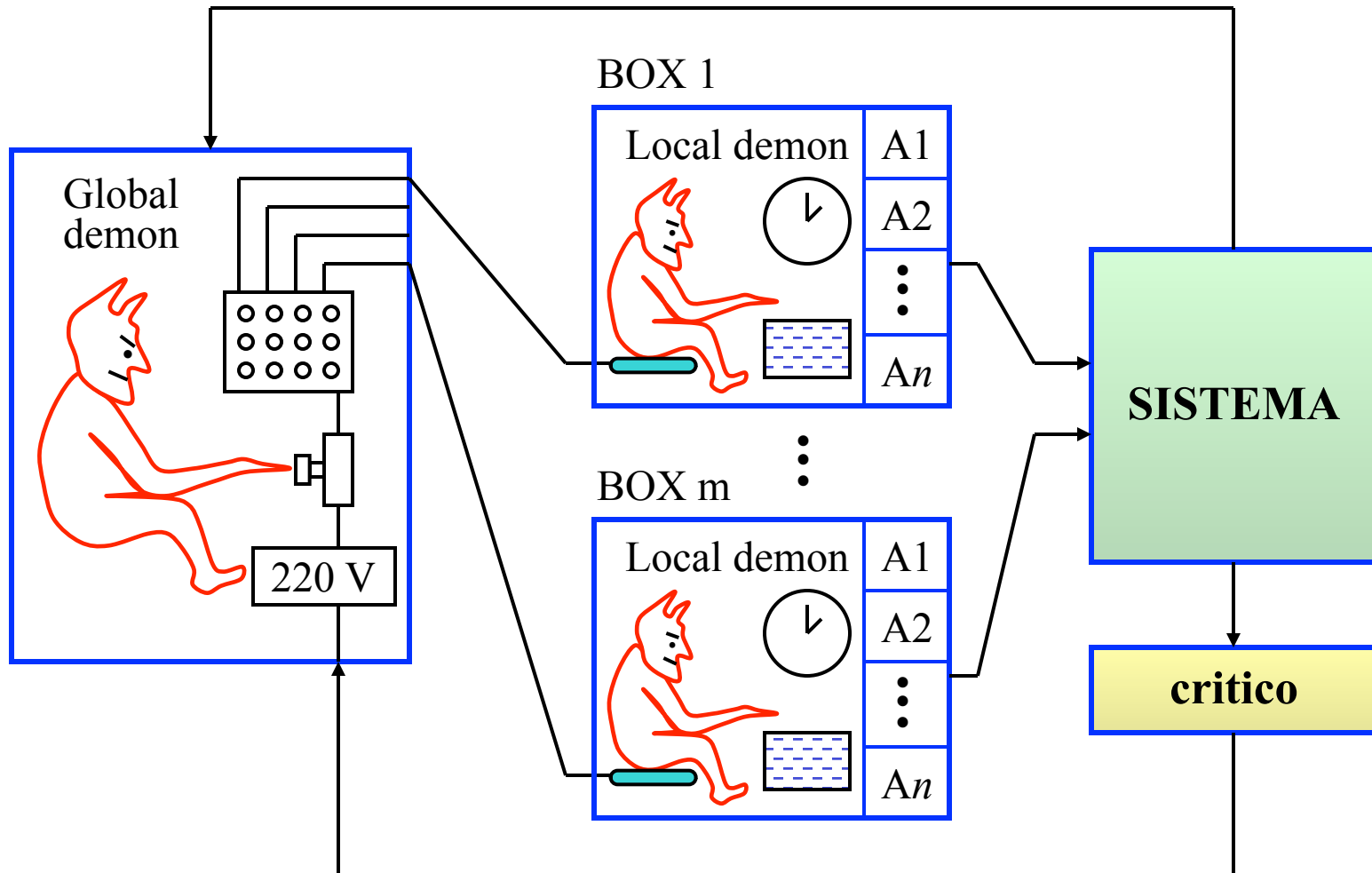
Un **agente** opera in un ambiente e modifica le azioni in base alle conseguenze prodotte.



Modello a scatole (Michie/Chamber '68)

- Apprendimento basato su punizioni.
- Si quantizza lo spazio degli stati in N regioni disgiunte (box).
- Ogni volta che il sistema entra in uno stato (box) viene scelta un'azione di controllo.
- Il controllore deve imparare a fare l'azione che più ritarda la punizione.

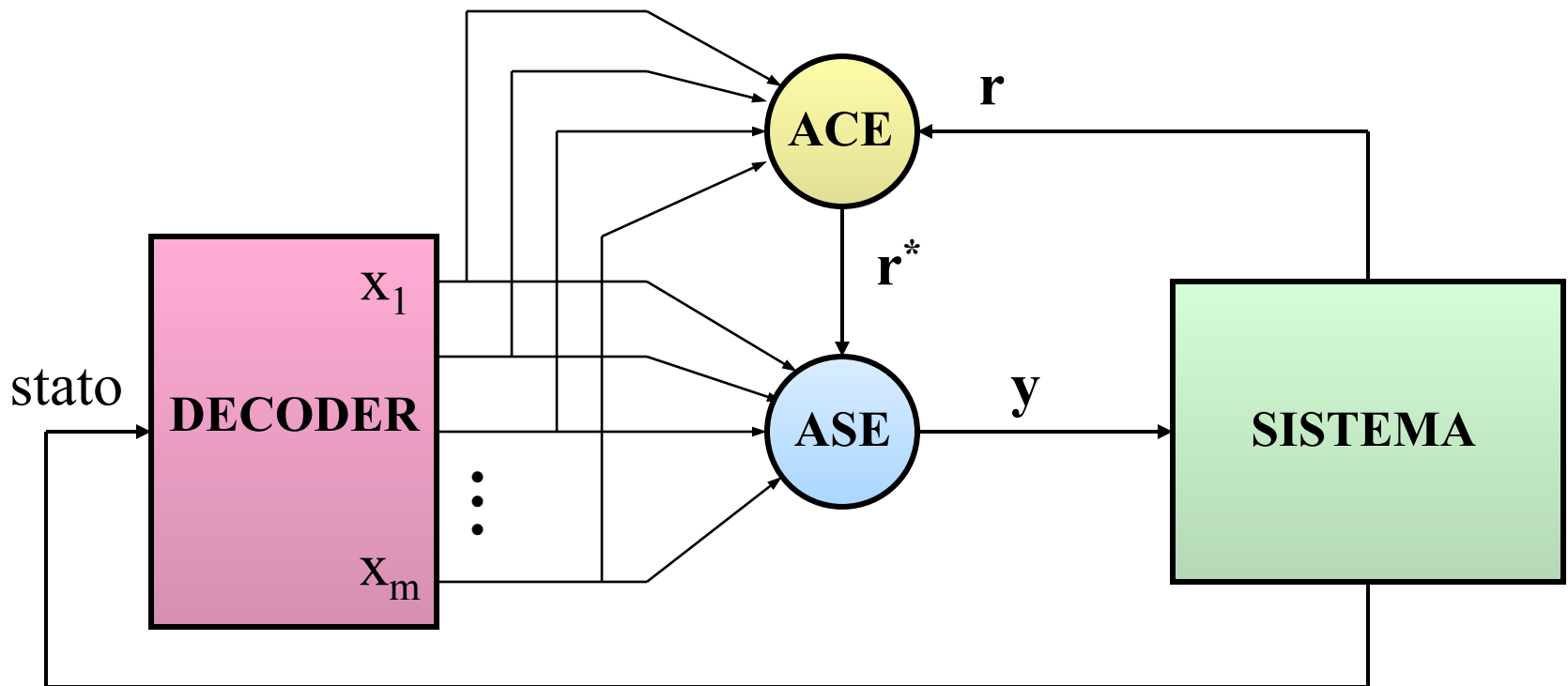
Modello a scatole



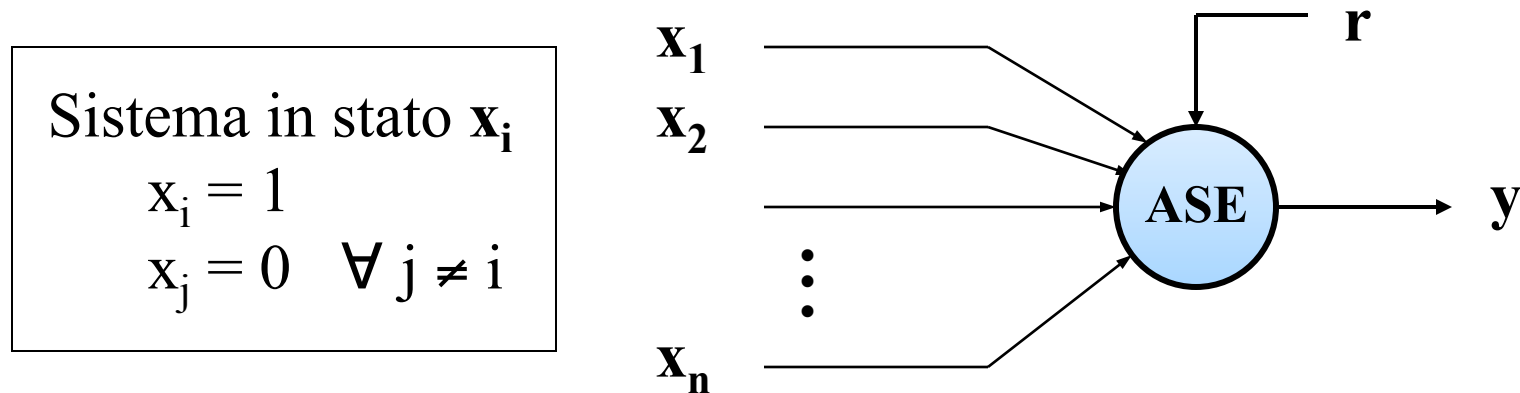
Modello neurale: ASE-ACE

(Barto-Sutton-Anderson, '83)

ASE: Associative Search Element
ACE: Adaptive Critic Element



Adaptive Search Element



$$y(t) = \text{sgn} \left[\sum_{i=1}^n w_i x_i + n(t) \right]$$

$\mathbf{n}(t)$ è una variabile gaussiana con
valor medio nullo e varianza σ^2

Adaptive Search Element

$$y(t) = \operatorname{sgn} \left[\sum_{i=1}^n w_i x_i + n(t) \right]$$

L' ASE può generare solo due azioni: $\begin{cases} \mathbf{a}^+ & (y = 1) \\ \mathbf{a}^- & (y = 0) \end{cases}$

Quando il sistema è nello stato \mathbf{x}_i ($\mathbf{x}_i = 1$) si ha che:

- se $\mathbf{w}_i = 0$, le azioni \mathbf{a}^+ e \mathbf{a}^- sono equiprobabili
- se $\mathbf{w}_i > 0$, \mathbf{a}^+ è scelta con maggiore probabilità
- se $\mathbf{w}_i < 0$, \mathbf{a}^- è scelta con maggiore probabilità

Adaptive Search Element

I pesi dell' ASE sono aggiornati con la seguente legge:

$$\Delta w_i(t) = \alpha r(t) e_i(t)$$

α è una costante positiva (**learning rate**)

$r(t)$ è il segnale di rinforzo

$$r(t) = \begin{cases} -1 & \text{in caso di fallimento} \\ 0 & \text{altrimenti} \end{cases}$$

$e_i(t)$ è un segnale (**elegibilità**) che introduce una memoria a breve termine sulle sinapsi:

$$e_i(t+1) = \delta e_i(t) + (1 - \delta) y(t) x_i(t)$$

Adaptive Search Element

$$\Delta w_i(t) = \alpha r(t) e_i(t)$$

$$e_i(t+1) = \delta e_i(t) + (1 - \delta) y(t) x_i(t)$$

Un fallimento ($r < 0$) riduce la probabilità di scelta delle azioni recenti che lo hanno causato:

$$a^+ \Rightarrow e_i(t) > 0 \Rightarrow \Delta w_i < 0$$

$$a^- \Rightarrow e_i(t) < 0 \Rightarrow \Delta w_i > 0$$

Adaptive Search Element

$$\Delta w_i(t) = \alpha r(t) e_i(t)$$

$$e_i(t+1) = \delta e_i(t) + (1 - \delta) y(t) x_i(t)$$

Un successo ($r > 0$) aumenta la probabilità di scelta delle azioni recenti che lo hanno causato:

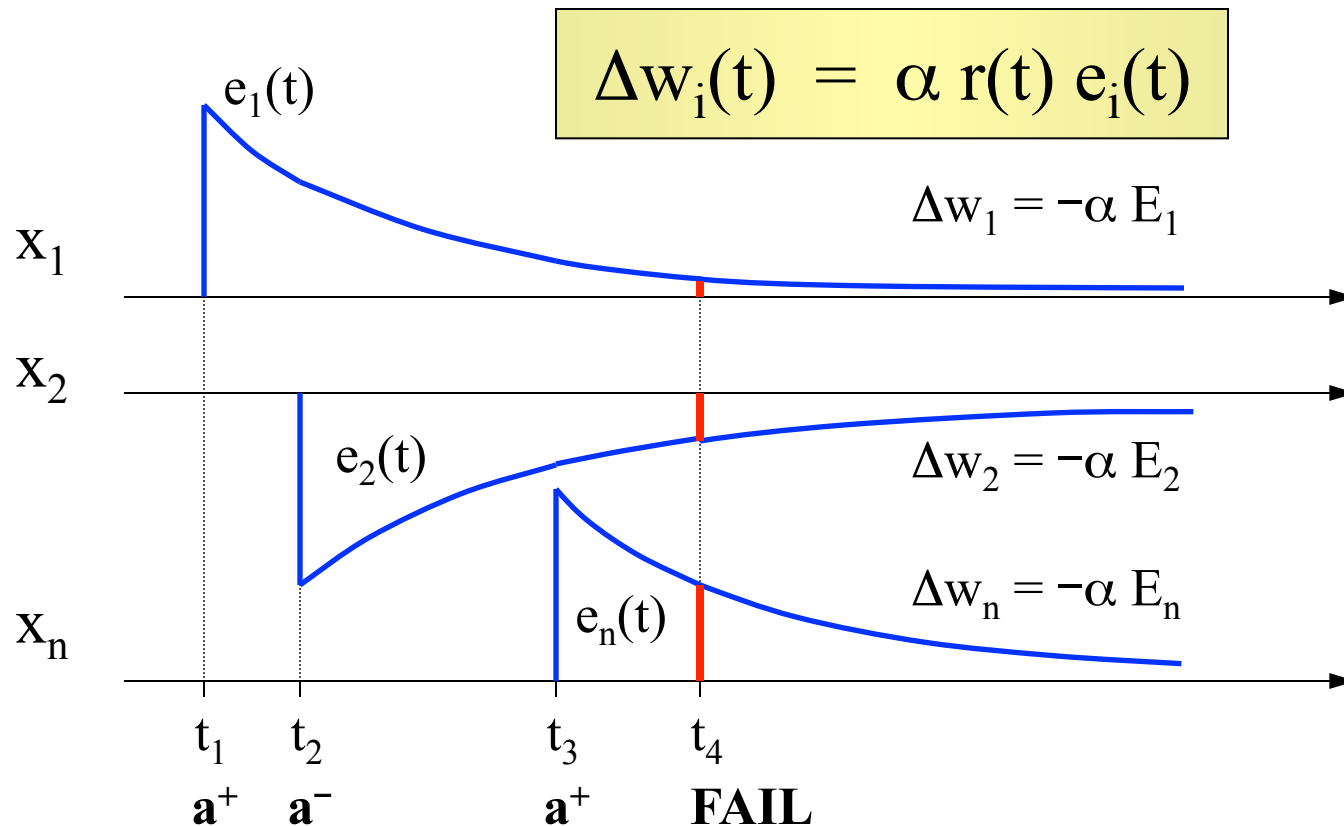
$$a^+ \Rightarrow e_i(t) > 0 \Rightarrow \Delta w_i > 0$$

$$a^- \Rightarrow e_i(t) < 0 \Rightarrow \Delta w_i < 0$$

Elegibilità

$$e_i(t+1) = \delta e_i(t) + (1 - \delta) y(t) x_i(t)$$

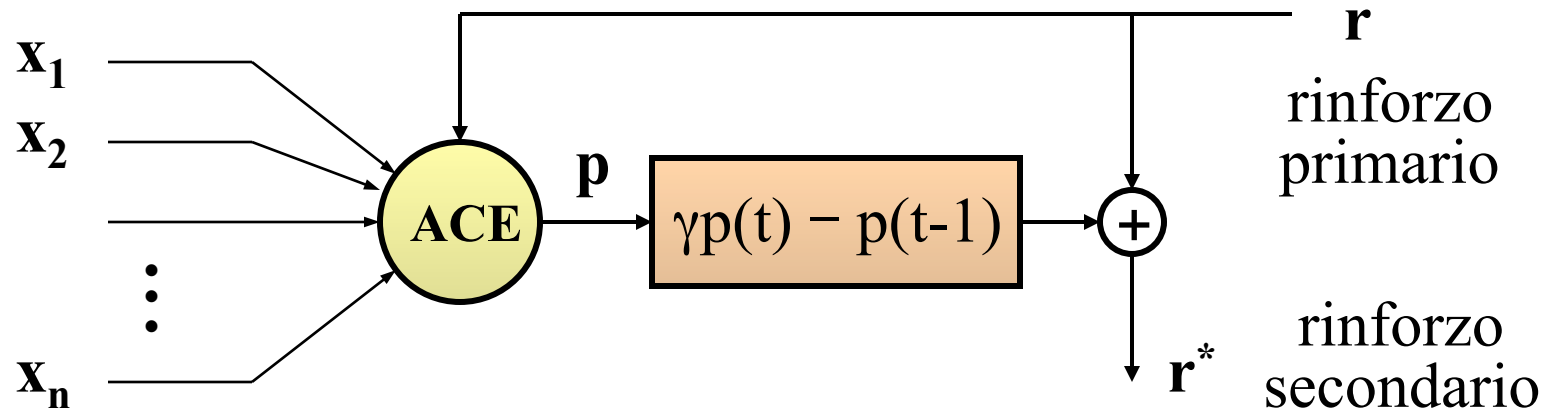
$$\Delta w_i(t) = \alpha r(t) e_i(t)$$



Importanza dell' ACE

- Man mano che i fallimenti si riducono, il sistema tenderebbe ad imparare più lentamente.
- Allora si introduce un critico adattivo (**ACE**), che genera un **rinforzo secondario** più informativo.
- Osservando lo stato del sistema e i fallimenti, l' ACE impara a prevedere gli stati pericolosi.
- Esso genera un premio ($r^* > 0$) se il sistema si allontana da uno stato pericoloso; una punizione ($r^* < 0$) in caso contrario.

Adaptive Critic Element



- L' ACE è addestrato in modo che $p(t)$ converga verso una previsione del rinforzo primario.
- Quindi, r^* viene generato in funzione della variazione della previsione.

Rinforzo secondario

$$r^*(t) = r(t) + \gamma p(t) - p(t-1)$$

- Se ($r = 0$) incrementi della previsione si traducono in premi ($r^* > 0$), decrementi in punizioni ($r^* < 0$).
- Se ($r = -1$), $x_i = 0 \forall i$, quindi $p(t) = 0$. Dunque:
 - Se il fallimento è stato predetto [$p(t-1) = r(t)$] si ha $r^* = 0$, tuttavia le azioni recenti sono state già punite negli istanti precedenti.
 - Se il fallimento non è stato predetto [$p(t-1) = 0$] si ha $r^* < 0$, che comunque genera punizione.