

# Introduzione alle reti neurali ed ai neurocontrollori

# Storia

- **Artificial Neural Networks** (ANNs) sono una simulazione astratta del nostro sistema nervoso, che contiene una collezione di neuroni i quali comunicano fra loro mediante connessioni dette *assoni*.
- Il modello ANN ha una certa somiglianza con gli assoni e dendriti in un sistema nervoso.
- Il primo modello di reti neurali fu proposto nel 1943 da McCulloch e Pitts nei termini di un modello computazionale dell'attività nervosa. A questo modello sono seguiti altri proposti da John von Neumann, Marvin Minsky, Frank Rosenblatt, e molti altri.

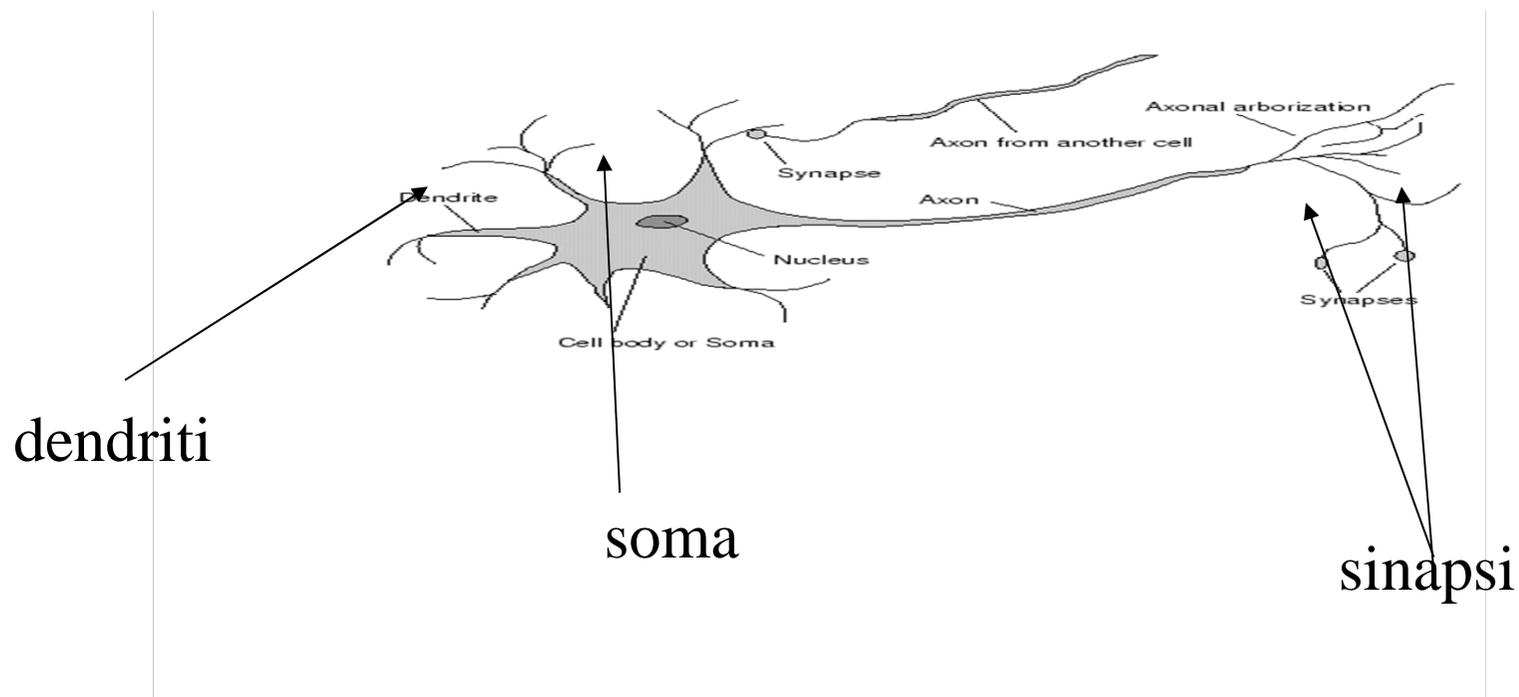
# Due categorie di modelli

- La prima è il "tipo biologico". Ha l'obiettivo di imitare sistemi neurali biologici, come le funzionalità auditive e visive. L'obiettivo principale di questo tipo di reti è la verifica di ipotesi riguardo ai sistemi biologici
- Il secondo tipo è guidato dalle applicazioni. E' meno interessato a "mimare" funzioni biologiche. Le architetture sono qui ampiamente condizionate dalle necessità applicative. Questi modelli sono anche denominati "**architetture connessioniste**".

Qui ci occuperemo del secondo tipo di reti.

# Neuroni

Molti neuroni posseggono strutture arboree chiamate **dendriti** che ricevono segnali da altri neuroni mediante giunzioni dette **sinapsi**. Alcuni neuroni comunicano mediante poche sinapsi, altri ne posseggono migliaia.



# Funzionamento di un neurone:

- Si stima che il cervello umano contenga oltre **100 miliardi di neuroni**. Studi sull'anatomia del cervello indicano che un neurone può avere oltre 1000 sinapsi in ingresso e uscita.
- Benché il tempo di commutazione di un neurone sia di pochi **millisecondi**, dunque assai più lento di una porta logica, tuttavia esso ha una connettività centinaia di volte superiore
- In genere, un neurone trasmette "informazione" agli altri neuroni attraverso il proprio **assone**. Un assone trasmette informazione mediante impulsi elettrici, che dipendono da suo potenziale. L'informazione trasmessa può essere **eccitatoria o inibitoria**.
- Un neurone riceve in ingresso alle sue sinapsi segnali di varia natura, che vengono sommati.
- Se l'influenza eccitatoria è predominante, il neurone si attiva e genera messaggi informativi attraverso le sinapsi di uscita.

# Struttura delle Reti

Una rete neurale è costituita da:

- Un insieme di nodi (**neuroni**), o unità connesse da collegamenti.
- Un insieme di **pesi** associati ai collegamenti.
- Un insieme di **soglie** o livelli di attivazione.

La **progettazione** di una rete neurale richiede:

1. La scelta del numero e del tipo di unità.
2. La determinazione della struttura morfologica.
3. Codifica degli esempi di addestramento, in termini di ingressi e di uscite della rete.
4. L'inizializzazione e l'addestramento dei pesi sulle interconnessioni, attraverso l'insieme di esempi di learning.

# Problemi risolvibili con le reti neurali

Caratteristiche:

- Le istanze sono rappresentate mediante molte features con molti valori, anche reali.
- La funzione obiettivo può essere a valori discreti, continui, o un vettore con attributi di tipo "misto"
- Gli esempi possono essere rumorosi
- Tempi di apprendimento lunghi sono accettabili
- La valutazione della rete "appresa" deve essere effettuata velocemente
- **Non è cruciale "capire" la semantica della funzione appresa**

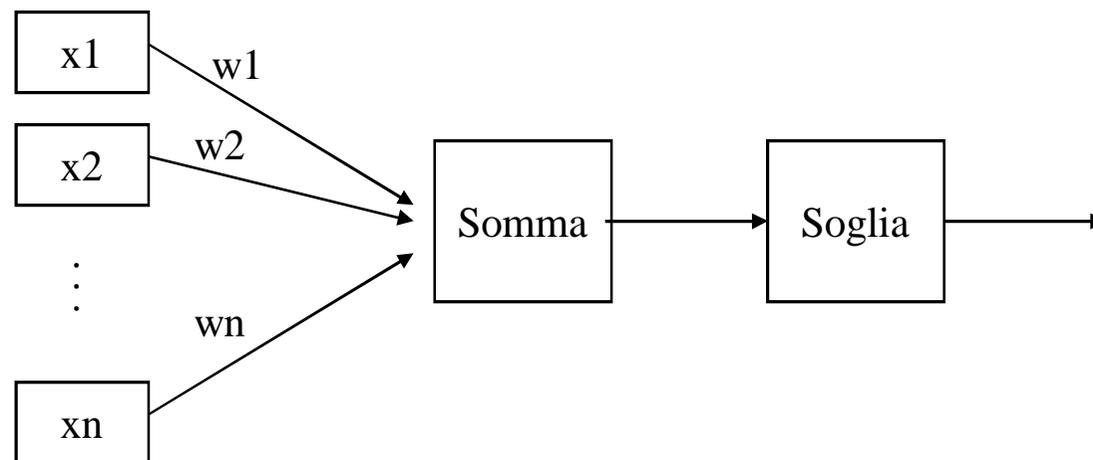
**Robotica, Image Understanding, Biological Systems**

# Apprendimento

- Supervised Learning
  - MLP e reti neurali ricorrenti
  - RBF
- Unsupervised Learning
  - Clustering
- Competitive Learning
  - Kohonen networks
- Reinforcement Learning

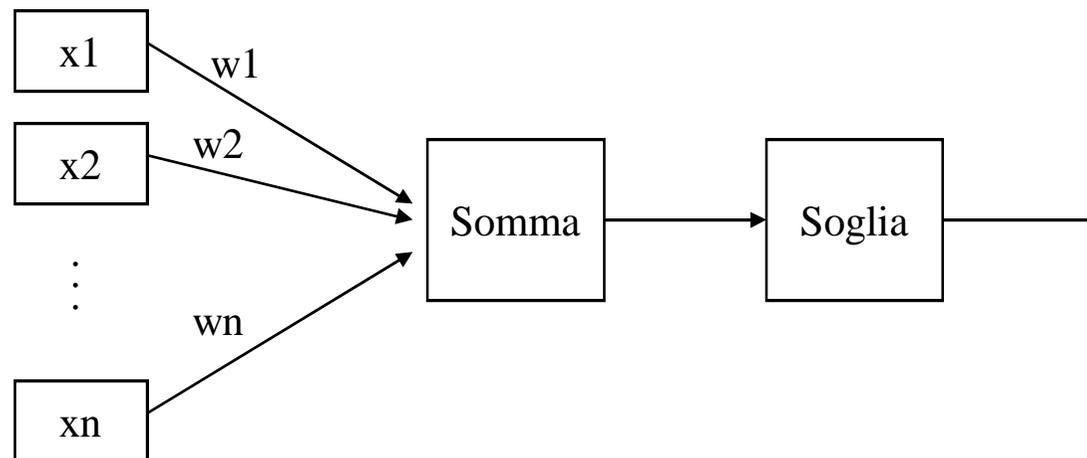
# Il Percettrone

- Il percettrone è il mattone base delle reti neurali
- Nasce da un'idea di Rosenblatt (1962)
- Cerca di simulare il funzionamento del singolo neurone



# Il Percettrone

- I valori di uscita sono booleani: 0 oppure 1
- Gli ingressi  $x_i$  e i pesi  $w_i$  sono valori reali positivi o negativi
- Ingressi, somma, soglia:
- L'apprendimento consiste nel selezionare pesi e soglia

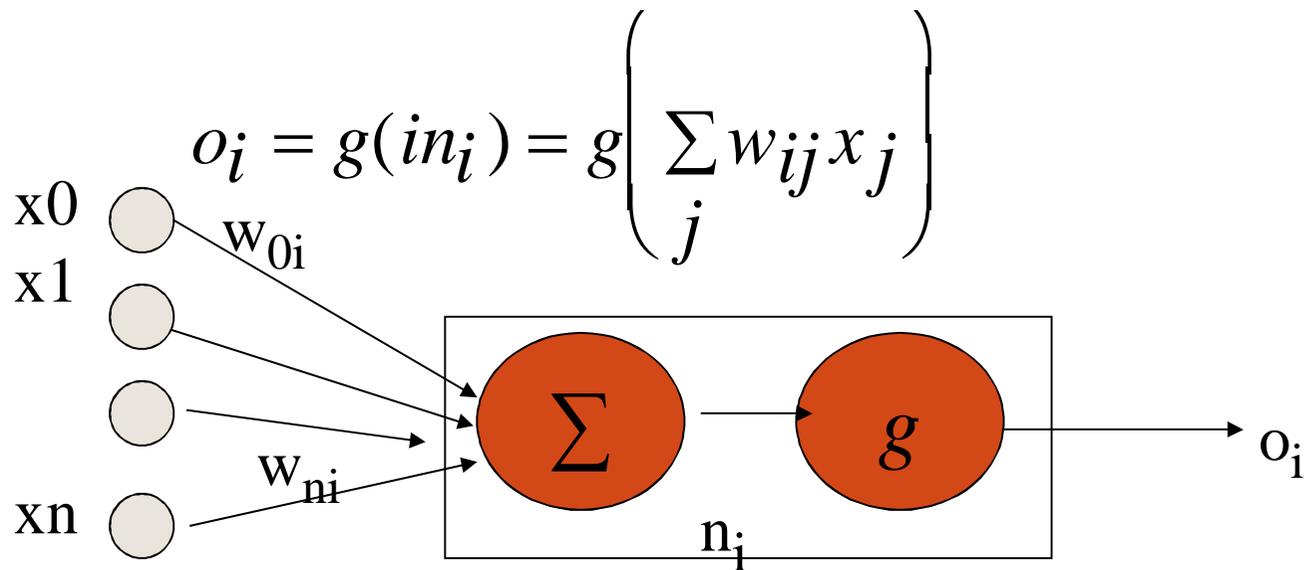


# Funzioni somma e soglia

a) **funzione d'ingresso**, *lineare (SOMMA)*

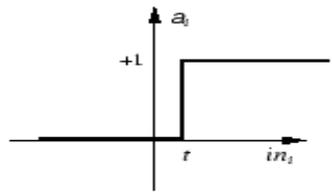
$$in_i = \sum_j w_{ij} x_j = w_i x_i$$

b) **funzione di attivazione**, *non lineare (SOGLIA)*

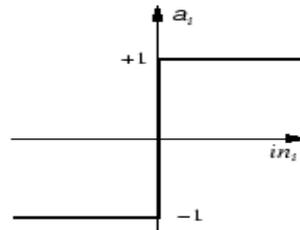


# Funzioni di attivazione

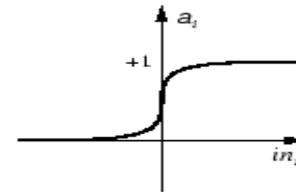
$$\text{gradino}_t(x) = \begin{cases} 1, & \text{se } x > t \\ 0, & \text{altrimenti} \end{cases}$$



(a) Step function



(b) Sign function



(c) Sigmoid function

$$\text{segno}(x) = \begin{cases} +1, & \text{se } x \geq 0 \\ -1, & \text{altrimenti} \end{cases}$$

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

# Funzione obiettivo

- Se la soglia è la funzione *segno*, e  $x_1 \dots x_n$  sono i valori degli attributi delle istanze  $x$  di  $X$ , si ha:

$$o(x) = 1 \text{ se } w_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n > 0$$

$$o(x) = -1 \text{ altrimenti}$$

- Esprimibile anche in forma vettoriale mediante la:

$$o(\dot{x}) = \text{sign}(\dot{w} \cdot \dot{x})$$

# Il Percettrone (classificazione e generalizzazione)

- Problema di apprendimento:
  - dati insiemi di punti su uno spazio n-dimensionale, classificarli in due gruppi (positivi e negativi)
  - inoltre dato un nuovo punto P decidere a quale gruppo appartiene
- Il primo problema è di classificazione, mentre per risolvere il secondo è richiesta capacità di generalizzazione, come all'apprendimento di concetti;

# Problema di classificazione

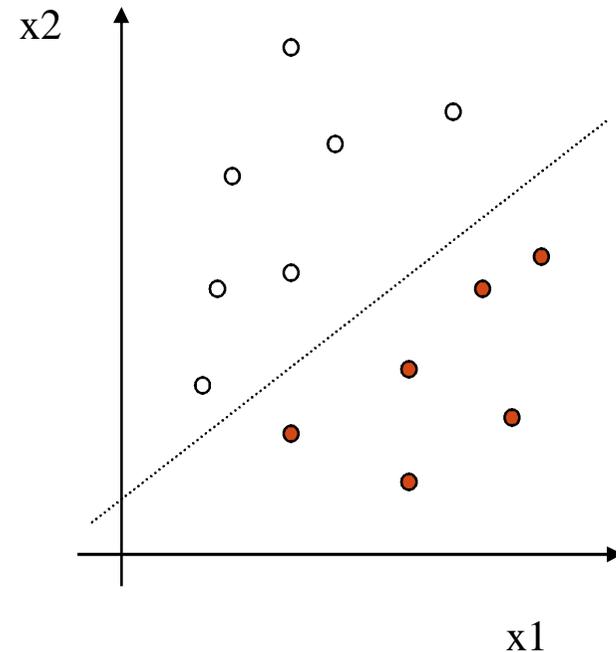
- Il problema è quindi ridotto alla determinazione dell'insieme dei pesi  $(w_0, w_1, \dots, w_n)$  migliore per minimizzare gli errori di classificazione
- Quindi lo spazio delle ipotesi  $H$  è infinito
- Si tratta di ottimizzare la funzione

$$H = \{ \overset{\rho}{\underset{\cdot}{w}} : \overset{\rho}{\underset{\cdot}{w}} \in \mathfrak{R}^{n+1} \}$$

$$o(\overset{\cdot}{x}) = \text{sign}(\overset{\cdot}{w} \cdot \overset{\cdot}{x})$$

# Esempio per due attributi

- Con  $(x_1, x_2)$  in ingresso, si ha:  
$$o(x) = w_0 + w_1 x_1 + w_2 x_2$$
mentre l'uscita è data da:  
1 se  $o(x) > 0$   
0 se  $o(x) < 0$
- La retta di separazione è data da:  
$$x_2 = - (w_1/w_2) x_1 - (w_0/w_2)$$
- Nel caso con  $n$  attributi, quel che si apprende è un *iperpiano di separazione*



# Algoritmo di addestramento del perceptrone

- Inizializza i pesi casualmente
- Sottoponi un esempio  $\langle x, c(x) \rangle$  di D
- Calcola la funzione  $o(x)$
- Se  $o(x) \neq c(x)$  allora aggiorna:
- $\eta$  si chiama *learning rate*
- $x_i$  è il valore dell'attributo i-esimo di  $x$
- La quantità  $(c-o)$  rappresenta l'errore E del perceptrone

$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = \eta(c(x) - o(x))x_i$$

# Esempio

- Supponiamo  $o(x) = -1$  (se la funzione soglia è  $\text{sign}(x)$ ) e  $c(x) = 1$
- Bisogna modificare i pesi per accrescere il valore di
- Se per esempio:

$$x_i = 0,8, \quad \eta = 0,1, \quad c = 1, \quad o = -1 \qquad \dot{w} \cdot \dot{x}$$

$$\Delta w_i = \eta(c - o)x_i = 0,1(1 - (-1))0,8 = 0,16$$

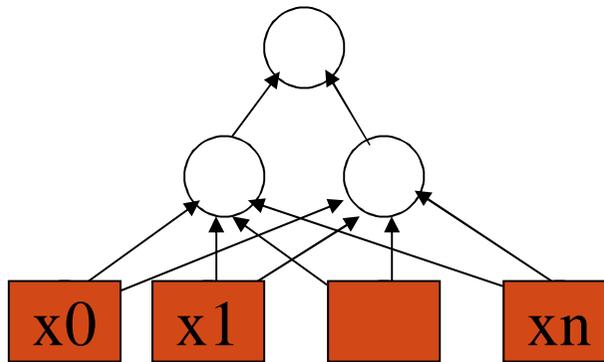
- Quindi il valore dei  $w_i$  tenderà a crescere, riducendo l'errore.
- Se invece  $c(x) = -1$  e  $o(x) = 1$

$$\Delta w_i = \eta(c - o)x_i = 0,1(-1 - (+1))0,8 = -0,16$$

# Il Percettrone

- Il *teorema di convergenza del percettrone* (Roseblatt, 1962) assicura che il percettrone riuscirà a delimitare le 2 classi se il sistema é linearmente separabile
- In altre parole, nell'ottimizzazione non esistono minimi locali
- Problema: come ci si comporta in caso di situazioni non linearmente separabili?
- Soluzioni: **reti multistrato alimentate in avanti, e reti ricorrenti**

# Reti alimentate in avanti

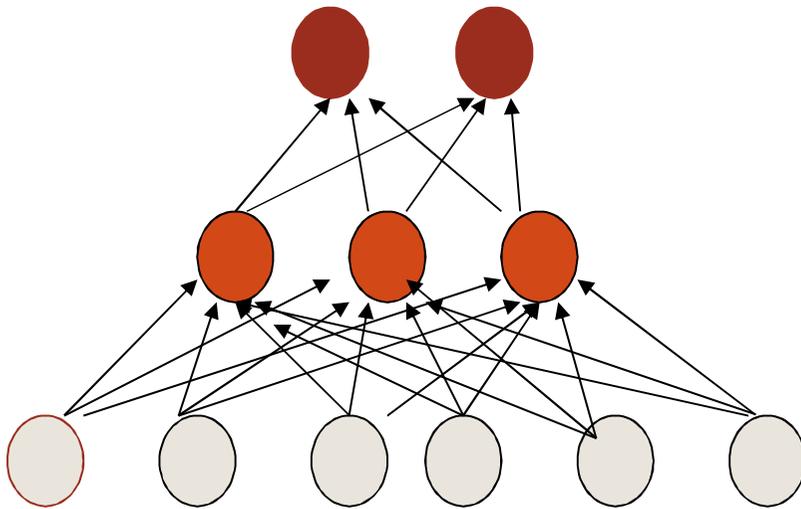


- Ogni unità è collegata solo a quella dello stato successivo
  - L'elaborazione procede uniformemente dalle unità di ingresso a quelle di uscita
  - Non c'è feedback
  - Non ha stato interno

# Reti Alimentate in Avanti : algoritmo con propagazione all'indietro (back propagation)

- Obiettivi:
  - partire da un insieme di percettroni con pesi casuali
  - apprendere in modo veloce
  - avere capacità di generalizzazione
  - avere insiemi di percettroni su larga scala

# Backpropagation (2)



-   $I_i$  Unità di ingresso
-   $H_j$  Unità nascoste
-   $O_k$  Unità di uscita

- La funzione soglia utilizzata è la sigmoide

$$o(x) = \sigma(w \cdot x) = \frac{1}{1 + e^{-w \cdot x}}$$

- La funzione di errore è calcolata come segue:

$$E(w) \equiv \frac{1}{2} \sum_{x \in D} (t(x) - o(x))^2 =$$

$$\frac{1}{2} \sum_{x \in D} \sum_{k \in N_{out}} (t_k(x) - o_k(x))^2$$

# Backpropagation (3)

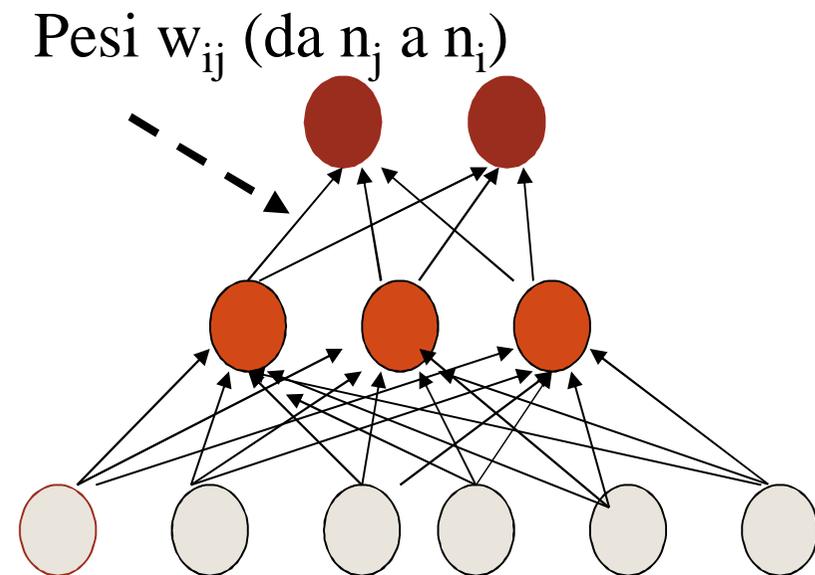
- Obiettivo: **minimizzare l'errore** fra ciascuna uscita desiderata e l'uscita calcolata dalla rete
- Regola di aggiornamento dei pesi:

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

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

$$\delta_j = o_j(1 - o_j)(t_j - o_j)$$

Nota: RN non generano, in generale, un solo output, ma **m** valori di output



# Backpropagation (4)

- D è un insieme di coppie  $(x, t(x))$ , dove  $x$  è il vettore dei valori di ingresso  $(x_1, x_2, \dots)$  e  $t$  è il vettore dei valori  $t_1, \dots, t_m$  della funzione obiettivo  $t$
- $\eta$  è il **learning rate**
- $x_{ji}$  rappresenta l'input dall'unità  $n_i$  all'unità  $n_j$  (e coincide con l'output di  $n_i$ ), mentre  $w_{ji}$  è il peso della connessione fra  $n_i$  e  $n_j$ .
- **Inizializzazione:**
  - Crea una architettura di rete  $G$  con  $N_{in}$  unità di ingresso,  $N_{out}$  unità di uscita,  $N_{hidden}$  unità nascoste
  - Inizializza tutti i pesi  $w_{ji}$  con valori **random**

# Backpropagation (5)

- **Finchè** non si raggiunge la condizione di terminazione, **esegui**:
- Per ogni esempio  $d \in D$ :  $(x, t(x))$  ( $x = (x_1, x_2, \dots, x_n)$ ,  $t(x) = (t_1, t_2, \dots, t_m)$ ):
  - Siano  $I$  i nodi di ingresso della rete  $(1, 2, \dots, n)$ ,  $O$  i nodi di uscita  $(1, 2, \dots, m)$ ,  $N$  l'insieme dei nodi della rete.
  - Poni in ingresso l'istanza  $x$  e calcola le uscite per ogni nodo  $n_u \in N$  della rete ( $x_i$  input del nodo di ingresso  $i \in I$ ,  $o_j$  output prodotto dal generico nodo  $n_j \in N$ )
  - Per ogni nodo di uscita  $o_k \in O$  calcola l'errore commesso, come segue:

$$\delta_k = o_k (1 - o_k) (t_k - o_k)$$

- Per ogni unità nascosta  $h_h \in H = (N - O \cup I)$  collegata ai nodi di  $O$  calcola l'errore commesso, come segue:  $\delta_h = o_h (1 - o_h) \sum_{n_k \in O} w_{kh} \delta_k$
- Calcola l'errore sui restanti nodi, procedendo all'indietro strato per strato
- Aggiorna tutti i pesi della rete come segue:

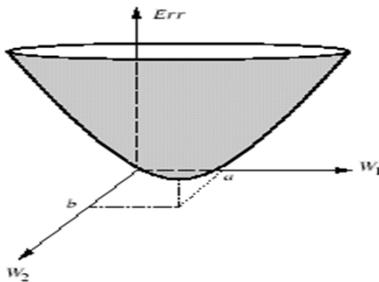
$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

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

# Spiegazione della regola di propagazione dell'errore all'indietro

Consideriamo la funzione di errore per  $x \in D$ :

$$E_x = \frac{1}{2} \sum_i (t_i - o_i)^2$$



Supponiamo di dover imparare solo due pesi.

Il piano  $w_1$   $w_2$  rappresenta lo spazio delle ipotesi (pesi delle connessioni), il cono è la superficie d'errore.

Per minimizzare l'errore si deve calcolare la direzione della discesa più ripida lungo la superficie.

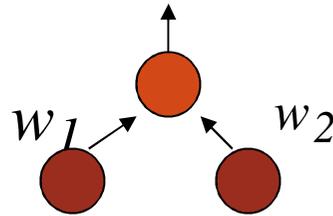
Quindi, la derivata.

# Esempio di calcolo del gradiente

$$\Delta w_1 = -\eta \frac{\partial E(w_1 x_1 + w_2 x_2)}{\partial w_1} = -\eta \frac{\partial E}{\partial net_1} \frac{\partial net_1}{\partial w_1} = -\eta \frac{\partial E}{\partial net_1} x_1$$

$$net_1 = w_1 x_1 + w_2 x_2$$

$$E = \frac{1}{2} (t - o)^2$$



$$\frac{\partial E}{\partial net_1} = \frac{\partial E}{\partial o} \frac{\partial o}{\partial net_1} \quad \frac{\partial E}{\partial o} = \frac{1}{2} 2(t - o) \frac{\partial (t - o)}{\partial o} = -(t - o)$$

$$\frac{\partial o}{\partial net_1} = \frac{\partial \sigma(net_1)}{\partial net_1} = o(1 - o)$$

$$\partial(\sigma(x)) = \sigma(x)(1 - \sigma(x))$$

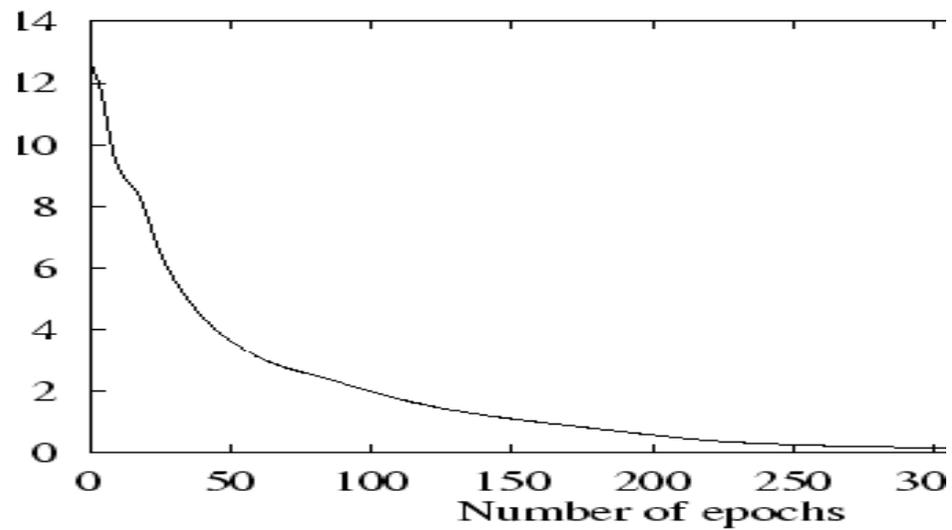
$$\Delta w_1 = \eta o(1 - o)(t - o)x_1$$

È la formula  
usata dall'algoritmo  
BP!!

## Condizioni di terminazione

- Il processo continua finchè non sono esauriti tutti gli esempi (*epoca*), poi si riparte
- Quando arrestare il processo? Minimizzare gli errori sul set  $D$  non è un buon criterio (*overfitting*)
- Si preferisce minimizzare gli errori su un test set  $T$ , cioè suddividere  $D$  in  $D' \cup T$ , addestrare su  $D'$  e usare  $T$  per determinare la condizione di terminazione.

# Plot dell'errore su un training set T



# Ma l'algoritmo converge?

- Problemi dell'algoritmo del gradiente:
  - Può arrestarsi su minimi locali
  - Un minimo locale può dare soluzioni molto peggiori del minimo globale
  - Possono esserci molti minimi locali
- Soluzioni possibili: addestra la rete con pesi iniziali diversi

# Modifica della regola di aggiornamento

- Quando viene osservato l'esempio n-esimo di D, la regola di aggiornamento diventa: 
$$\Delta w_{ij}(n) \leftarrow \eta \delta_j x_{ij} + \alpha \Delta w_{ij}(n-1)$$
- Il secondo termine prende il nome di *momento*.
- Vantaggi:
  - È possibile superare minimi locali
  - Mantiene i pesi nelle zone dove l'errore è piatto
  - Aumenta la velocità dove il gradiente non cambia
- Svantaggi:
  - Se il momento è troppo alto, si può cadere in massimi locali
  - E' un parametro in più da regolare!

## Alcune considerazioni pratiche

- La scelta dei pesi iniziali ha un grande impatto sul problema della convergenza! Se la dimensione dei vettori di input è  $N$  ed  $N$  è grande, una buona euristica è scegliere i pesi iniziali fra  $-1/N$  e  $1/N$
- L'algoritmo BP è molto sensibile al fattore di apprendimento  $\eta$ . Se è troppo grande, la rete diverge.
- A volte, è preferibile usare diversi valori di  $\eta$  per i diversi strati della rete
- La scelta della modalità di codifica degli ingressi e della architettura  $G$  della rete può influenzare in modo drastico le prestazioni!!

# Alcune considerazioni pratiche (2)

- **Struttura ottimale della rete:**

Come tutti i modelli statistici, anche le reti neurali sono soggette a sovradattamento. In questo caso, il sovradattamento può essere causato da una struttura di rete non efficiente, troppo "piccola" o troppo "grande".

Non esiste alcuna buona teoria per caratterizzare le funzioni rappresentabili efficientemente tramite reti!

(una possibile risposta è rappresentata dagli **algoritmi genetici**)

- **Espressività:**

Non hanno il potere espressivo delle rappresentazioni logiche (come gli alberi di decisione). Ma a differenza di questi ultimi, sono adatte a rappresentare funzioni per ingressi e uscite di tipo continuo. In termini molto generici, sono adatte per funzioni per le quali le interazioni fra ingressi non sono "intricate" e l'uscita varia gradualmente al variare degli ingressi. Inoltre, tollerano bene il rumore.

- **Efficienza computazionale:**

Per  $m$  esempi e  $|w|$  pesi, ogni epoca richiede un tempo  $O(m|w|)$ .

- **Trasparenza:** le reti neurali sono "scatole nere". Non hanno semantica!

# Overfitting, generalizzazione, criteri di stop

Un problema importante è capire quando arrestare il processo di iterazione (stopping criterion).

Osservare solo l'andamento dell'errore è una strategia non buona.

Al crescere delle iterazioni, il sistema cerca di adattarsi anche a esempi idiosincratici, generando una ipotesi molto complessa. Accade frequentemente che questa ipotesi perda di generalità, ovvero, commetta parecchi errori nel predire casi non visti.

Osservate come, sul test set, la probabilità di errore cresca, benché diminuisca sul training set.

Un buon metodo è la **cross-validation**.

I pesi con i valori migliori vengono usati per calcolare le prestazioni su un set di validazione, diverso da quello di apprendimento.

Il training termina quando i risultati cominciano a divergere.

# Reti Neurali Ricorrenti

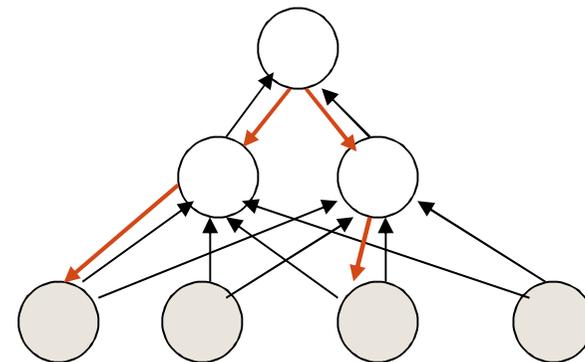
Sono un modello migliore del funzionamento del cervello umano: le reti alimentate in avanti non simulano la memoria a breve termine. Nel cervello, esistono evidenti connessioni all'indietro.

Dunque:

- I collegamenti possono formare configurazioni arbitrarie.
- L'attivazione viene "ripassata" indietro alle unità che l'hanno provocata
- Hanno uno stato interno: livelli di attivazione
- Computazione meno ordinata
- Instabili
- Più tempo per calcolare lo stato stabile
- Difficoltà nel learning
- Implementano agenti più complessi.

Esempi

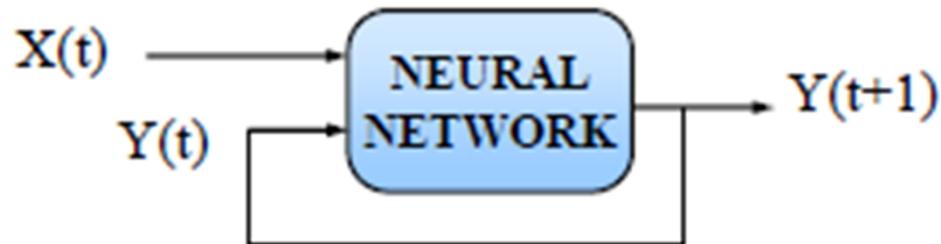
- Macchine di Boltzmann
- Reti di Hopfield



## Reti Neurali Ricorrenti (2)

- Sono reti neurali che imparano l'associazione tra un pattern di input e una serie di pattern di output

$$X_k \Rightarrow Y_{k1}, Y_{k2}, \dots, Y_{kL}$$



Le connessioni tra le unità formano cicli. Si crea uno stato interno della rete che gli permette un comportamento temporale dinamico.

# Apprendimento

- Supervised Learning
  - MLP e reti neurali ricorrenti
  - RBF
- Unsupervised Learning
  - Clustering
- Competitive Learning
  - Kohonen networks
- Reinforcement Learning

# Apprendimento non supervisionato

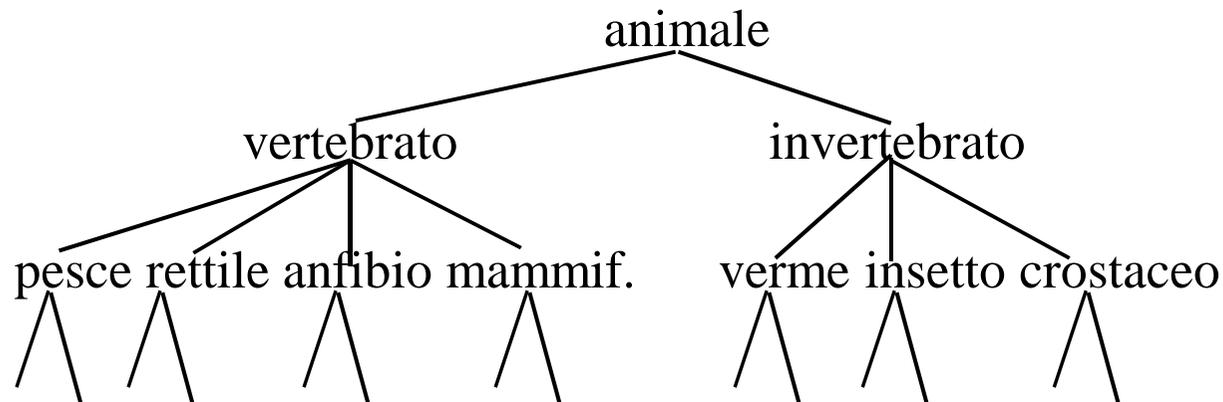
- Suddivide esempi non etichettati in sottoinsiemi disgiunti (**cluster**), tali che:
  - Gli esempi in uno stesso gruppo sono “molto” simili
  - Gli esempi in gruppi diversi sono “molto” differenti
- Scopre **nuove categorie** in modo **non supervisionato** (a priori non vengono fornite etichette per le categorie)

# Tipi di Clustering

- Clustering **gerarchico** (hierarchical clustering)
  - Formano cluster iterativamente utilizzando cluster precedentemente costituiti
- Clustering **partitivo** (partitional clustering)
  - Crea una sola partizione degli esempi in cluster minimizzando una certa funzione di costo

# Clustering Gerarchico

- Costruisce una gerarchia ad albero a partire da un insieme di esempi non etichettati



- **L'applicazione ricorsiva** di un algoritmo di clustering può produrre un clustering gerarchico
- Distinguiamo due tipi di clustering gerarchico:
  - Agglomerativo (bottom-up)
  - Divisivo (top-down)

# Clustering Partitivo

- Si deve fornire il numero desiderato di cluster  $k$
- Si scelgono  $k$  istanze a caso, una per cluster, chiamate **semi (seeds)**
  - Si formano i  $k$  cluster iniziali sulla base dei semi
- Itera, riallocando tutte le istanze sui diversi cluster per migliorare il clustering complessivo
- Ci si ferma quando il clustering converge o dopo un numero prefissato di iterazioni

# K-means

- Assume istanze a valori reali
- I cluster sono basati su **centroidi** o media dei punti in un

cluster  $c$ :

$$\mu(c) = \frac{1}{|c|} \sum_{x \in c} x$$

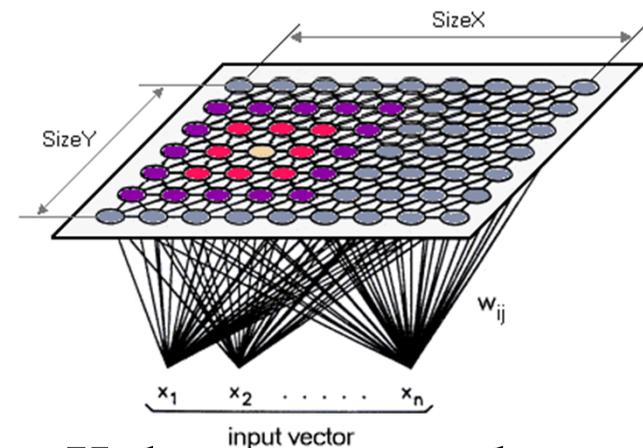
- Le istanze vengono riassegnate ai cluster sulla base della distanza rispetto ai centroidi dei cluster attuali

# Apprendimento

- Supervised Learning
  - MLP e reti neurali ricorrenti
  - RBF
- Unsupervised Learning
  - Clustering
- Competitive Learning
  - Reti di Kohonen
- Reinforcement Learning

# Apprendimento competitivo

- I neuroni competono per rispondere agli stimoli.
- Il neurone con l'output maggiore vince la competizione e si specializza nel riconoscere un dato stimolo
- Grazie a connessioni eccitatorie i neuroni vicini al vincitore sono sensibili a input simili



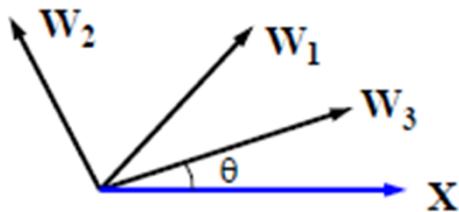
Kohonen network

Si crea un isomorfismo tra spazio di input e di output

# Implementazione

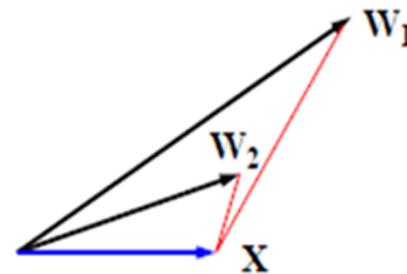
- Il neurone vincitore è selezionato usando una strategia globale comparando gli output degli altri neuroni.
- Due tecniche:
  - **1. Selezione il neurone con l'output maggiore;**
  - **2. Seleziona il neurone il cui vettore peso è più simile all'input**

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



**METHOD 1** – Il vincitore in caso di input  $X$  è quello con output maggiore

$$j_0 : \text{DIS}(W_{j_0}, X) < \text{DIS}(W_j, X) \quad \forall j \neq j_0$$



**METHOD 2** – Il neurone vincitore in caso di input  $X$  è quello che ha il vettore peso più simile a  $X$

# Apprendimento

- Supervised Learning
  - MLP e reti neurali ricorrenti
  - RBF
- Unsupervised Learning
  - Clustering
- Competitive Learning
  - Reti di Kohonen
- Reinforcement Learning

# Reinforcement learning

- Non sempre è possibile modellare un problema di apprendimento come la scelta ottimale di una funzione di classificazione
- L'apprendimento per rinforzo modella (tipicamente) il caso di un agente che percepisce ed agisce in un certo ambiente, il cui obiettivo è di imparare a fare "la cosa ottimale", o la cosa che lo avvicina di più al suo obiettivo, dato un certo stato dell'ambiente
- Robotica, web assistant, in generale, sistemi ad agenti

# Reinforcement learning: modello

Un agente si muove in un ambiente rappresentabile mediante un insieme  $S$  di stati;

L'agente è in grado di percepire un vettore di ingresso, o percezione  $e$ , che informa l'agente circa lo **stato**  $X_i$  in cui si trova.

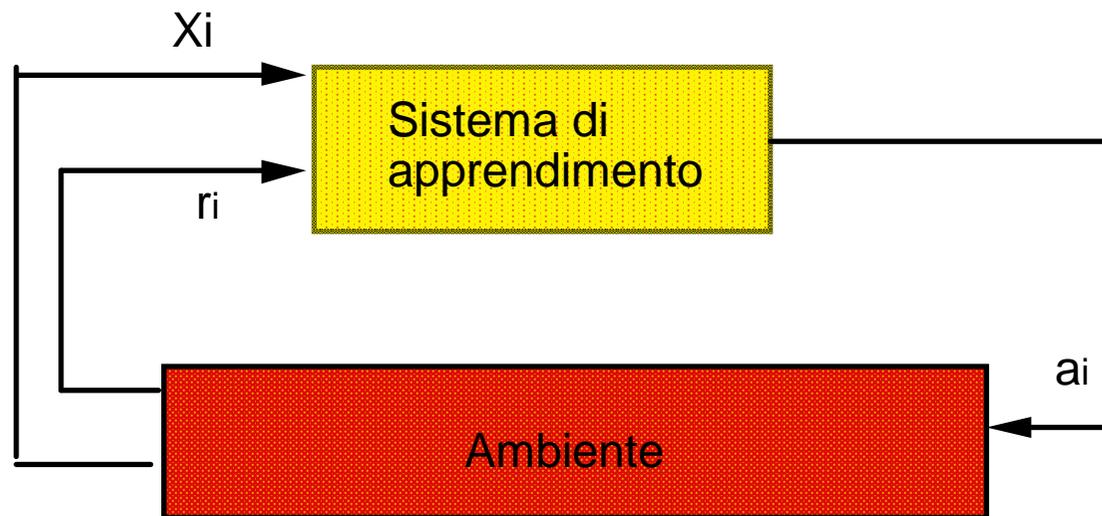
$A$  è un insieme di azioni eseguibili dall'agente.

L'esecuzione di un'azione  $a_i$  di  $A$  produce una transizione di stato.  $R$  è un insieme di **ricompense** che l'agente può ricevere:

$$r_i = r(e_i, a_i) \quad r_i \in R, \quad a_i \in A, \quad e_i = f(X_i),$$

## Modello (2)

L'obiettivo di apprendimento per rinforzo consiste nell'identificare una **politica ottima**  $\pi(X) X \rightarrow A$  che massimizzi le ricompense accumulate nel tempo.

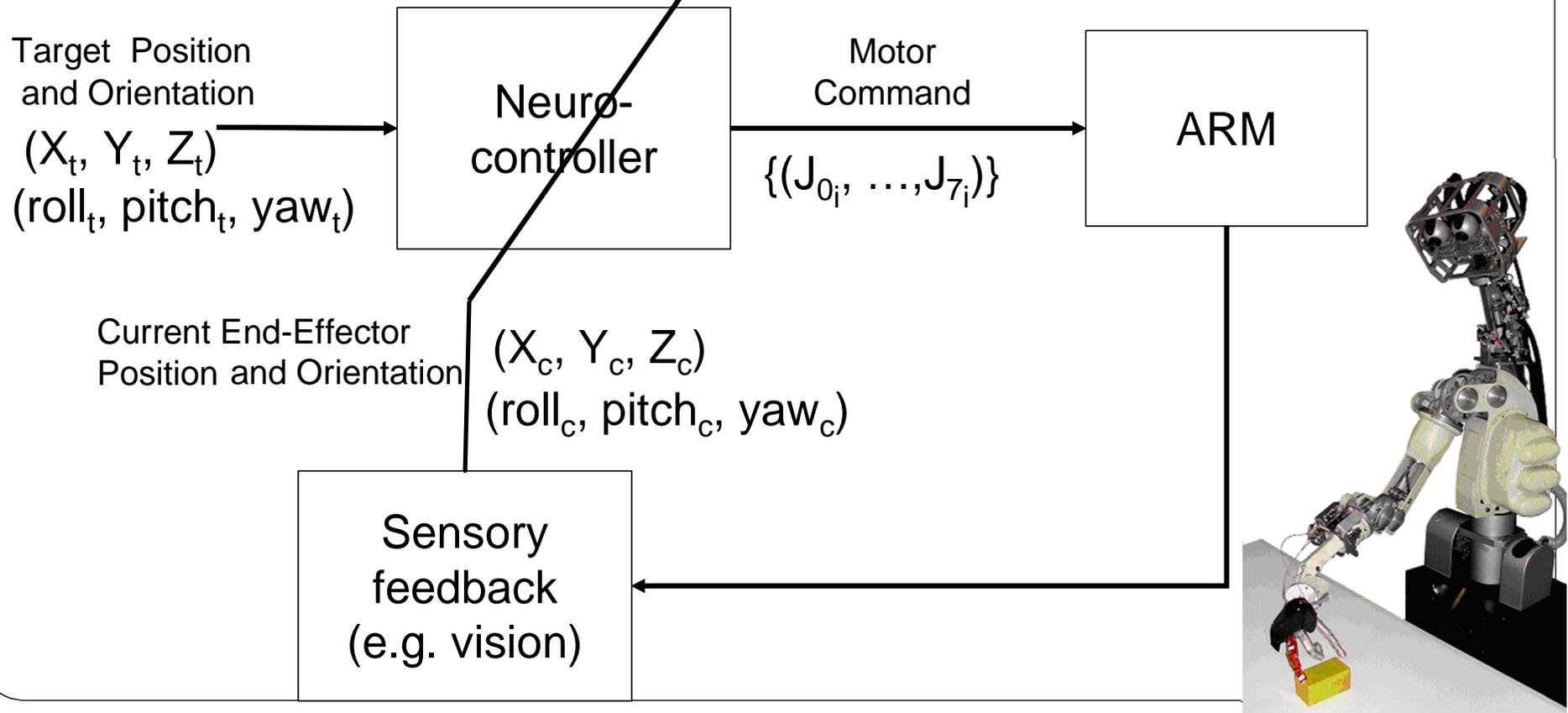


$$X_0 \xrightarrow{a_0} r_0 \quad X_1 \xrightarrow{a_1} r_1 \quad X_2 \xrightarrow{a_2} r_2 \quad \dots$$

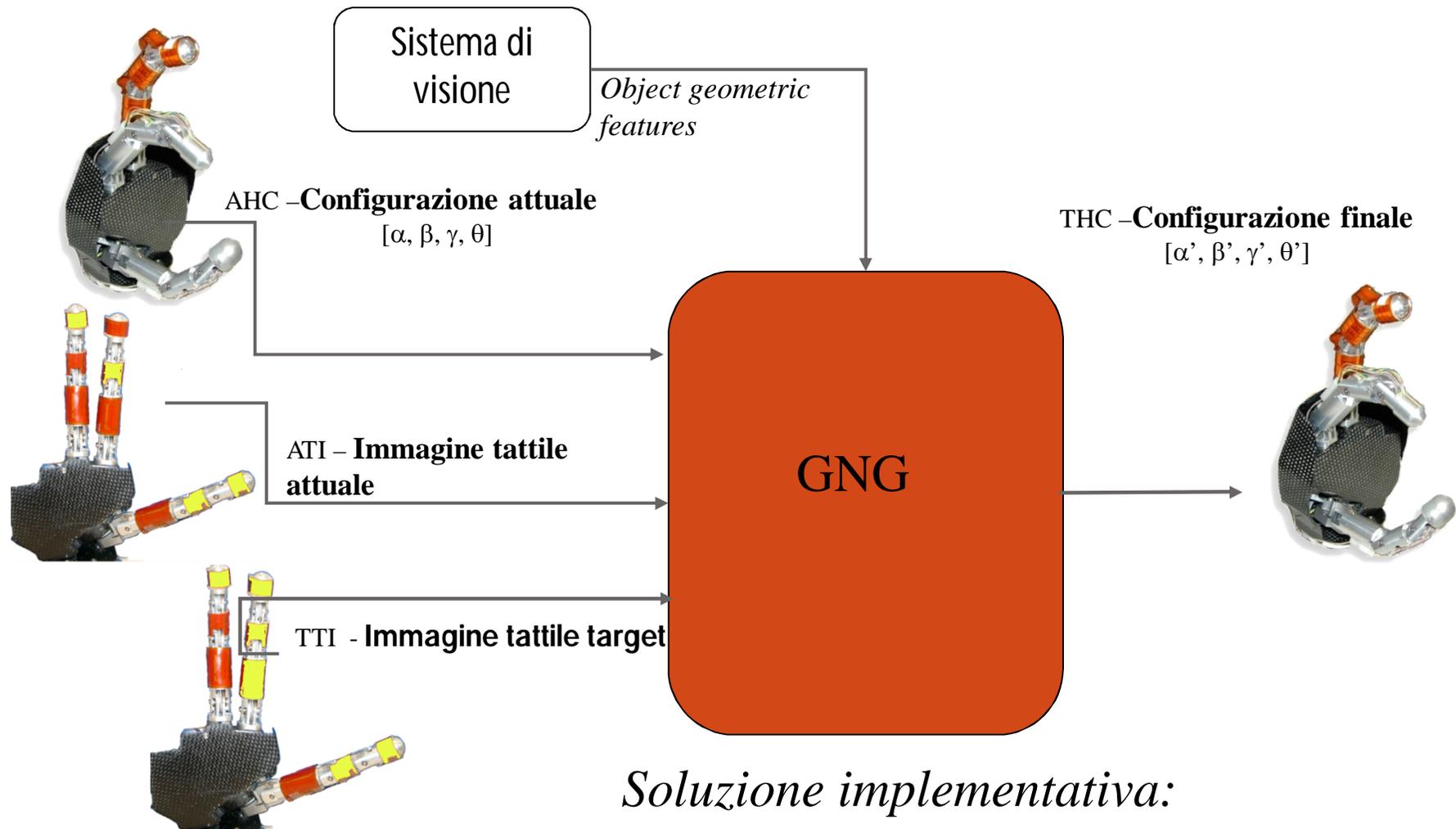
# Esempi di neurocontrollori

# 1) Un neurocontrollore per la posizione e l'orientazione del braccio robotico

- Nessuna conoscenza a priori della cinematica e della dinamica del robot
- **Capacità di learning**, per sviluppare un modello interno
- Bassa complessità computazionale



## 2) Modulo di grasping basato su EP



### 3) Modulo di anticipazione visiva basato su EP in task di locomozione

- Si basa sulla generazione di immagini 3D sintetiche predette (Percezione attesa)
- Riduzione del costo computazionale (la ricostruzione 3D dell'ambiente si effettua solo in caso di errore nel confronto tra immagine predetta e quella acquisita)
- Il modello interno è implementato attraverso una rete **neural-fuzzy**

## I concetti della logica fuzzy

Nella logica fuzzy ci si interessa di **INSIEMI FUZZY**, che corrispondono a **CATEGORIE** o **CONCETTI**

A differenza degli **INSIEMI USUALI**, in cui o un elemento appartiene all'insieme o non vi appartiene, negli insiemi fuzzy ogni elemento è associato ad un **GRADO DI APPARTENENZA**

Questo grado di appartenenza viene calcolato in base ad una opportuna **FUNZIONE DI APPARTENENZA**, caratteristica dell'insieme fuzzy considerato

## La funzione di appartenenza

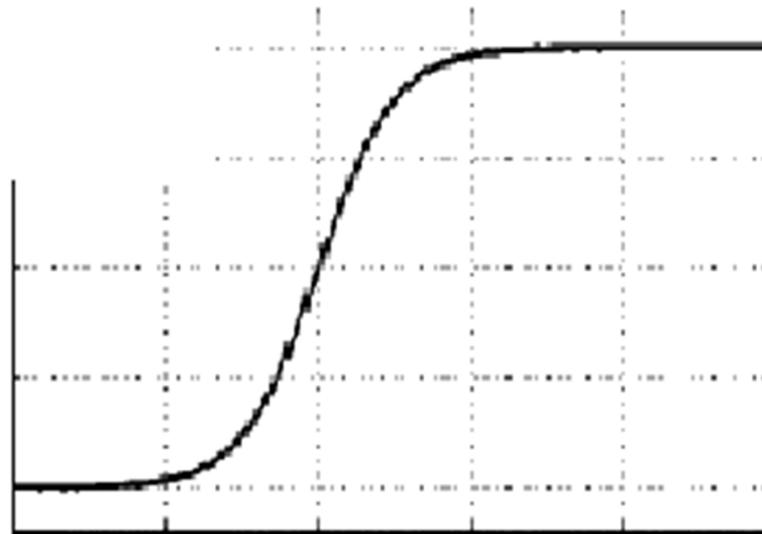
La funzione di appartenenza di un insieme fuzzy associa alle caratteristiche di ogni elemento (o ESEMPLARE o INDIVIDUO) il valore del grado di appartenenza all'insieme stesso

**Esistono molti tipi di funzioni di appartenenza.**

**Esse possono essere scelte per ragioni di pura convenienza matematica o determinate tramite osservazioni sperimentali**

## Un esempio

Grado di appartenenza



1.50

1.90

Altezza  
dell'individuo

**Funzione di appartenenza della categoria fuzzy  
"ABBASTANZA ALTO"**

# Le regole della logica Fuzzy

Usando i concetti fuzzy, si possono combinare tra loro con le regole della logica fuzzy per effettuare RAGIONAMENTI FUZZY

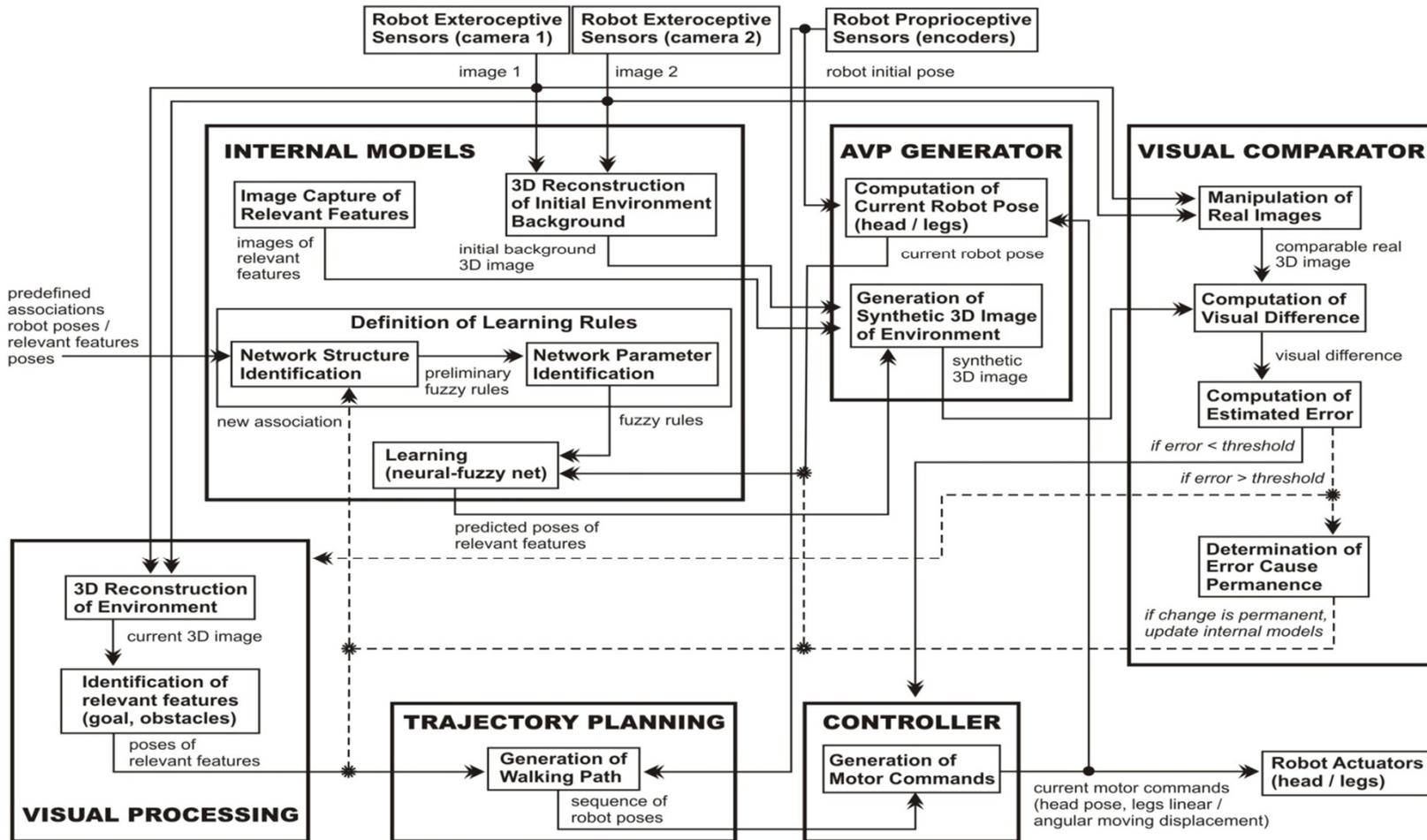
Ad esempio:

SE (x è ABBASTANZA ALTO) ALLORA (mostrargli un vestito di taglia L)

Il primo termine si chiama ANTECEDENTE e il secondo CONSEQUENTE

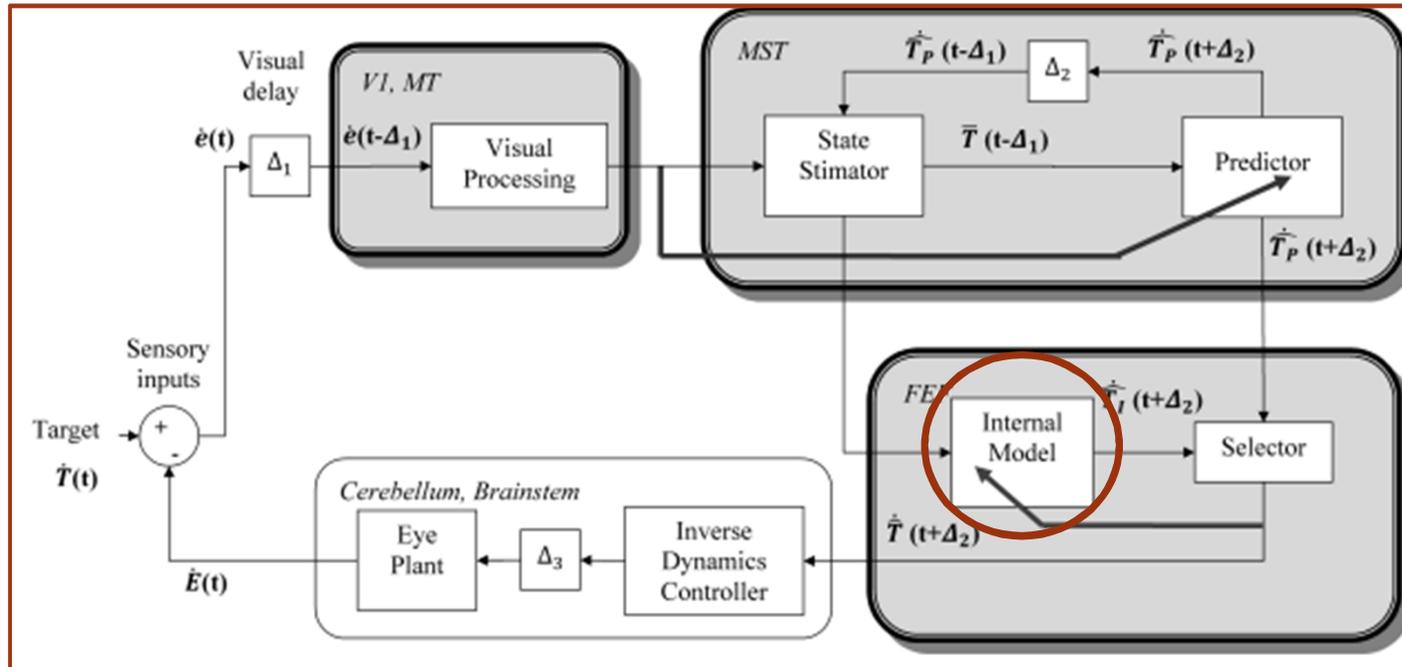
# Lo schema AVP

## THE AVP SCHEME



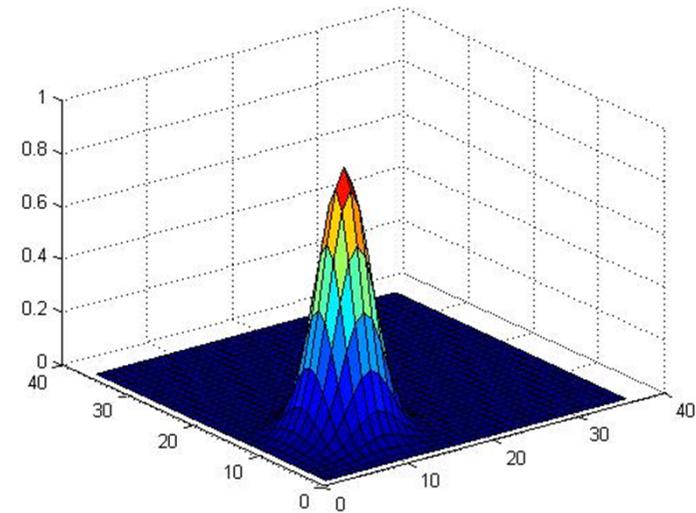
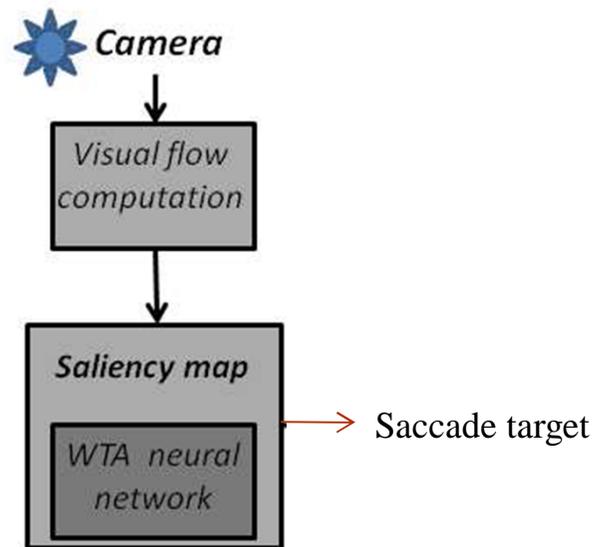
## 4) Rete neurale supervisionata per l'implementazione di modelli interni

- Smooth pursuit



Predizione della dinamica del target attraverso l'algorithmo RLS e una rete **MLP** che associ dinamiche conosciute con i pesi dell'algorithmo RLS

## 5) Competitive learning per la generazione di saliency map nella generazione di saccadi



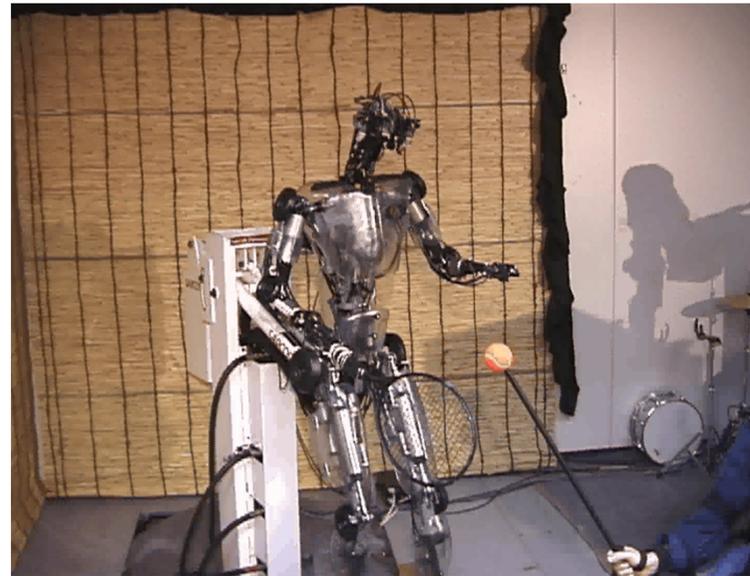
La rete neurale è utilizzata per modellare il processo di informazioni della corteccia cerebrale (Amari et al, 1977)

L'obiettivo è prendere in input stimoli spazialmente localizzati che competono per diventare il successivo punto target per il movimento oculare di saccadi

## 6 )Un Passo avanti: Learning from demonstration

- Apprendimento tramite imitazione può ridurre il costo nella programmazione di robot bioispirati
- Un movimento è mappato in un insieme infinito di movimenti primitivi che competono per l'azione percepita
- Ogni movimento primitivo predice l'uscita della percezione e prova a regolare i parametri per raggiungere una predizione sempre migliore, finché viene determinato il vincitore (**competitive learning**).

# Un Passo avanti: Learning from demonstration (II)



## **Reinforcement learning**

Tecniche di reinforcement learning sono usate dopo che il robot ha osservato la dimostrazione

Il robot acquisisce la funzione *reward* che gli permette di imparare dalla pratica senza ulteriori dimostrazioni.