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.

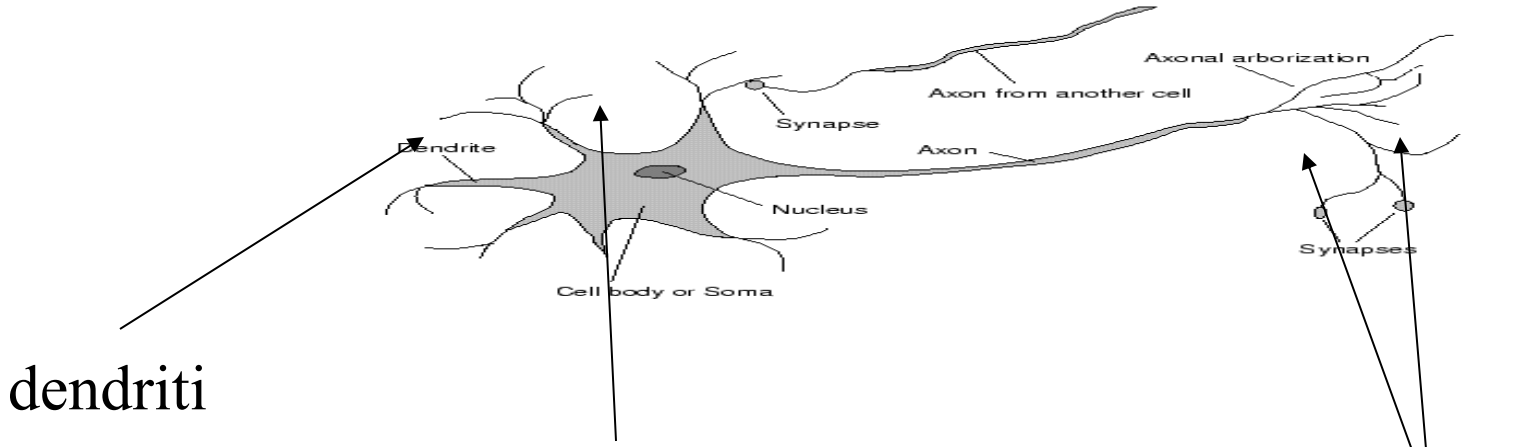
Due categorie di modelli

- 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.



dendriti

soma

sinapsi

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

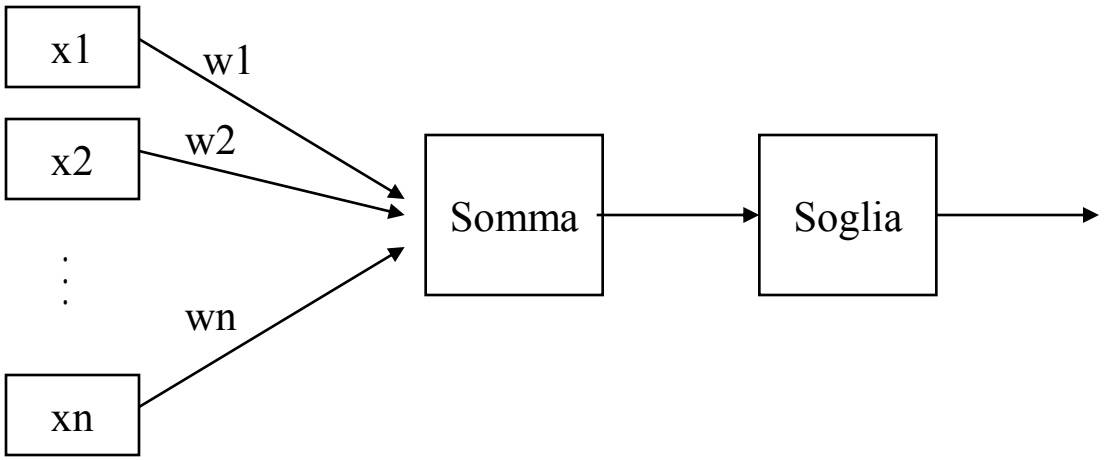
- Competitive Learning

  - Kohonen networks

- Reinforcement Learning

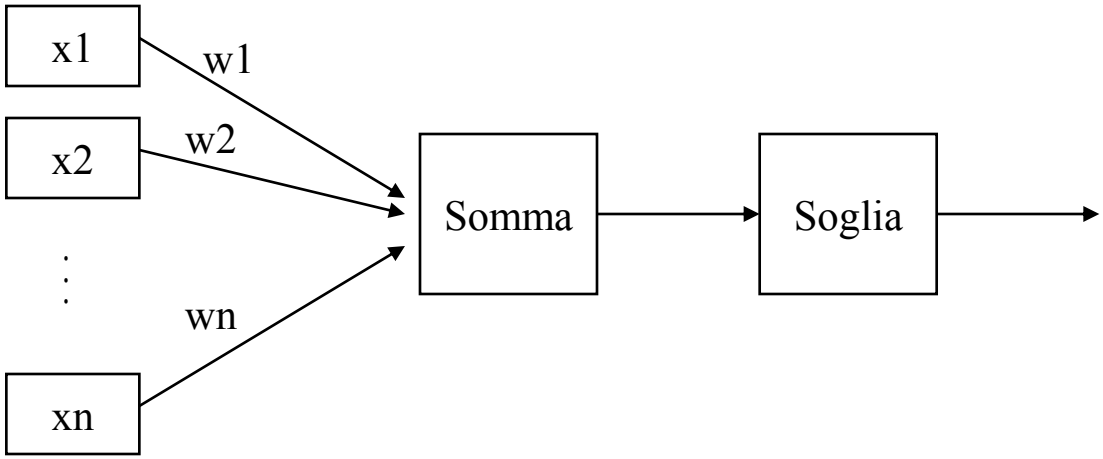
Il Percettrone

- 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 *xi* e i pesi *wi* sono valori reali positivi o negativi
- Ingressi, somma, soglia:
- L'apprendimento consiste nel selezionare pesi e soglia



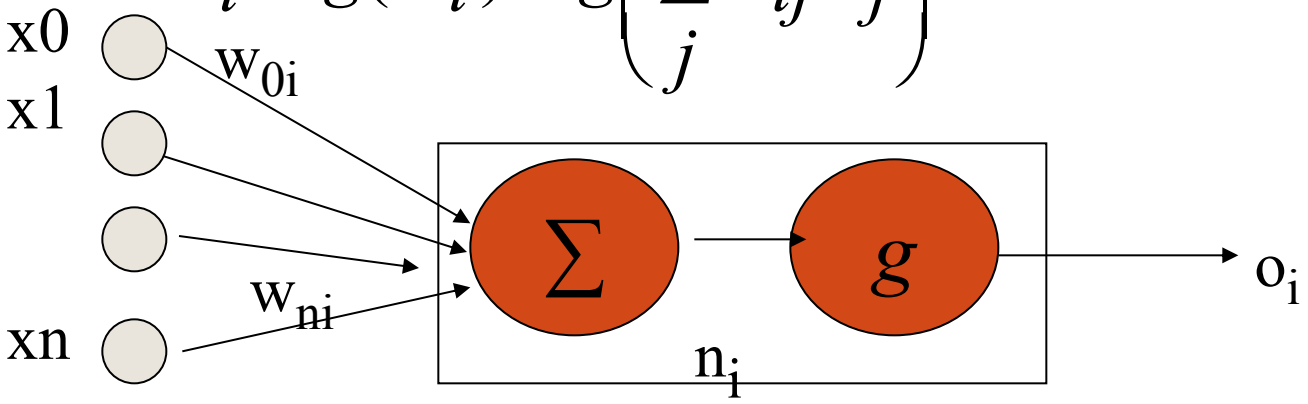
Funzioni somma e soglia

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

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

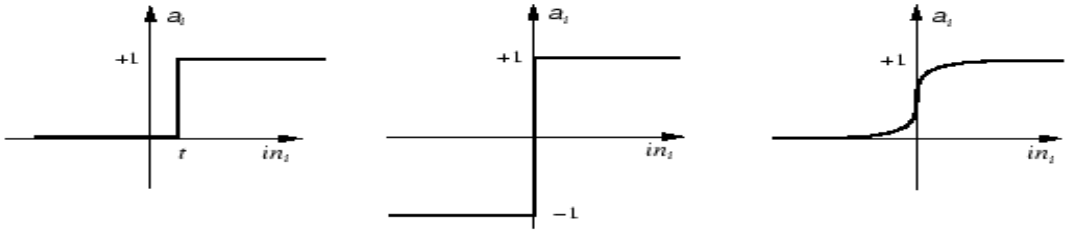
b) **funzione di attivazione**, *non lineare (SOGLIA)*

$$o_i = g(in_i) = g\left(\sum_j w_{ij} x_j\right)$$



Funzioni di attivazione

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



(a) Step function (b) Sign function (c) Sigmoid function

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

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

Funzione obiettivo

- Se la soglia è la funzione *segno*, e x1..xn sono i valori degli attributi delle istanze x di X, si ha:

$$o(x) = 1 \text{ se } w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > 0$$

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

- Esprimibile anche in froma vettoriale mediante la:

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

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 (w0, w1,… wn) migliore per minimizzare gli errori di classificazione
- Quindi lo spazio delle ipotesi H é infinito

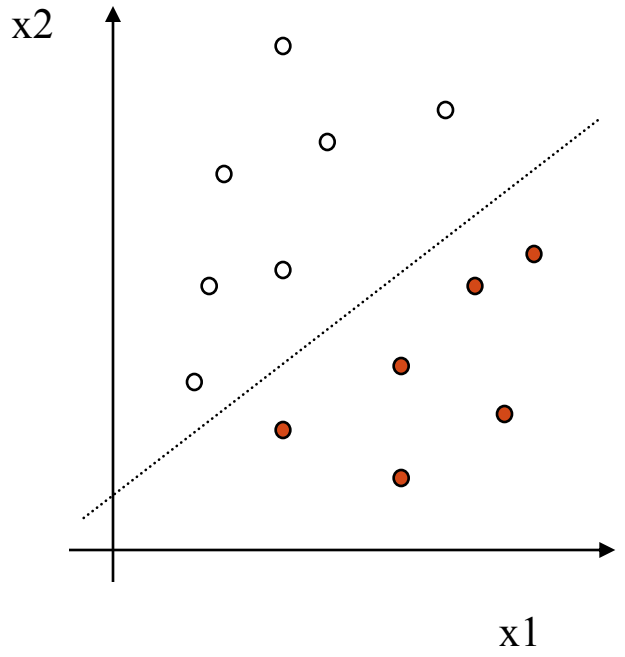
$$H = \left\{ \vec{w} : \vec{w} \in \Re^{n+1} \right\}$$

- Si tratta di ottimizzare la funzione

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

Esempio per due attributi

- Con (x1, x2) in ingresso,
  si ha:
  $o(x) = w0 + w1\ x1 + w2\ x2$
  mentre l'uscita è data da:
  1 se $o(x) > 0$
  0 se $o(x) < 0$
- La retta di separazione è data da:
  $x2 = -(w1/w2)\ x1 - (w0/w2)$
- Nel caso con n attributi, quel che si apprende è un *iperpiano di separazione*



Algoritmo di addestramento del percettrone

- Inizializza i pesi casualmente
- Sottoponi un esempio <x,c(x)> di D
- Calcola la funzione o(x)
- Se o(x)≠c(x) allora aggiorna:
- η si chiama *learning rate*
- x_i è il valore dell'attributo i-esimo di x
- La quantità (c-o) rappresenta l'errore E del percettrone

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

Esempio

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

$$x_i = 0,8, \quad \eta = 0,1, \quad c = 1, \quad o = -1$$

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

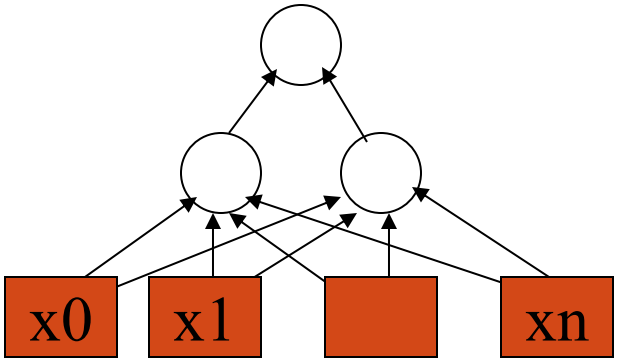
- Quindi il valore dei w_i tenderà a crescere, riducendo l'errore.
- Se invece c(x)=-1 e o(x)=1

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

Il Percettrone

- Il *teorema di convergenza del percettrone* (Rosemblatt, 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

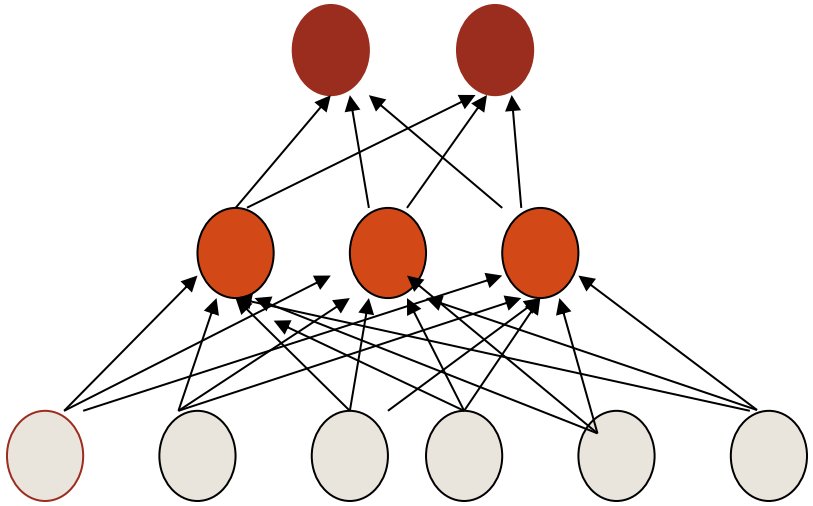


- Ogni unità è collegata solo a quella dello stato successivo
  - L'elaborazione procede uniformemente dalle unità di ingresso a quelle di uscita
  - Non c'è feedback
  - Non ha stato interno

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)



Ii Unità di ingresso
Hj Unità nascoste
Ok Unità di uscita

- La funzione soglia utilizzata è la sigmoide

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

- La funzione di errore è calcolata come segue:

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

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

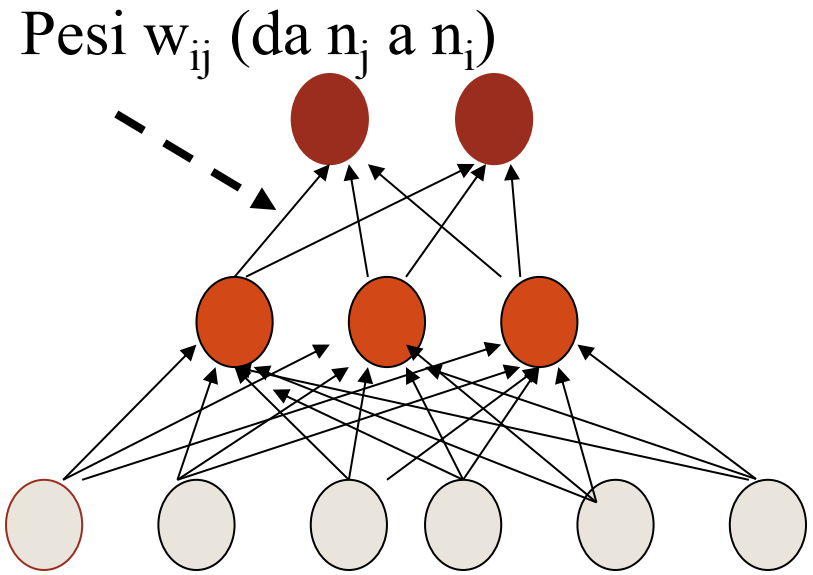
Backpropagation (3)

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

$$wji \leftarrow wji + \Delta wji$$

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

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



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

Backpropagation (4)

- **Finchè** non si raggiunge la condizione di terminazione, **esegui**:
- Per ogni esempio di∈D: (x, t(x)) (x=(x_1,x_2..x_n), t(x)=(t_1,t_2,..t_m):
  - Siano I i nodi di ingresso della rete (1,2,,n), O i nodi di uscita (1,2..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 $i_i \in I$, 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:

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

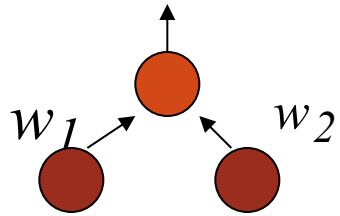
$$\Delta w_{ji} = \eta \delta_j x_{ji}$$

Esempio di calcolo del gradiente

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

$$net_1 = w_1 x_1 + w_2 x_2$$

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



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

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

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

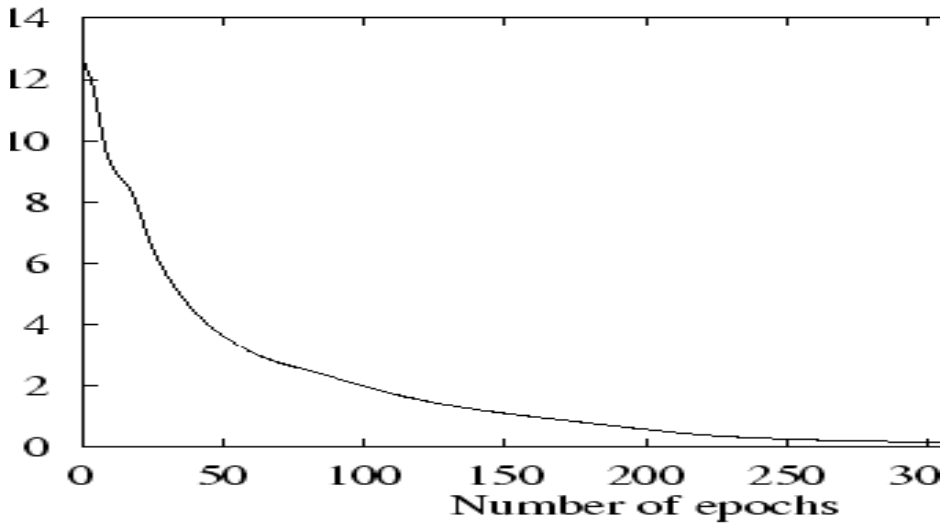
$$\Delta w_1 = \eta o(1 - o)(t - o) x_1$$

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: $\Delta wij(n) \leftarrow \eta \delta_j x_{ij} + \alpha \Delta w_{ij}(n-1)$
- 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

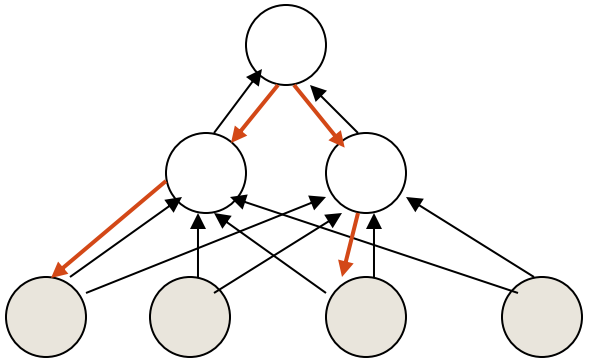
- La scelta dei pesi iniziali ha un grande impatto sul problema della convergenza! Se la dimensione dei vettori di input è N ed N è grande, una buona euristica è scegliere i pesi iniziali fra -1/N e 1/N
- L'algoritmo BP è molto sensibile al fattore di apprendimento η. Se è troppo grande, la rete diverge.
- A volte, è preferibile usare diversi valori di η 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!!

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.



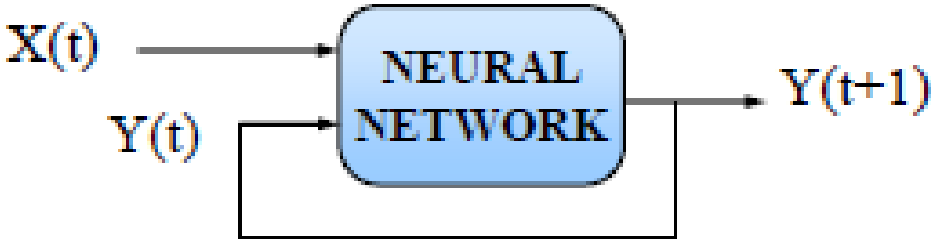
Esempi

- 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

$$Xk \Rightarrow Yk1, Yk2, \ldots, YkL$$



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

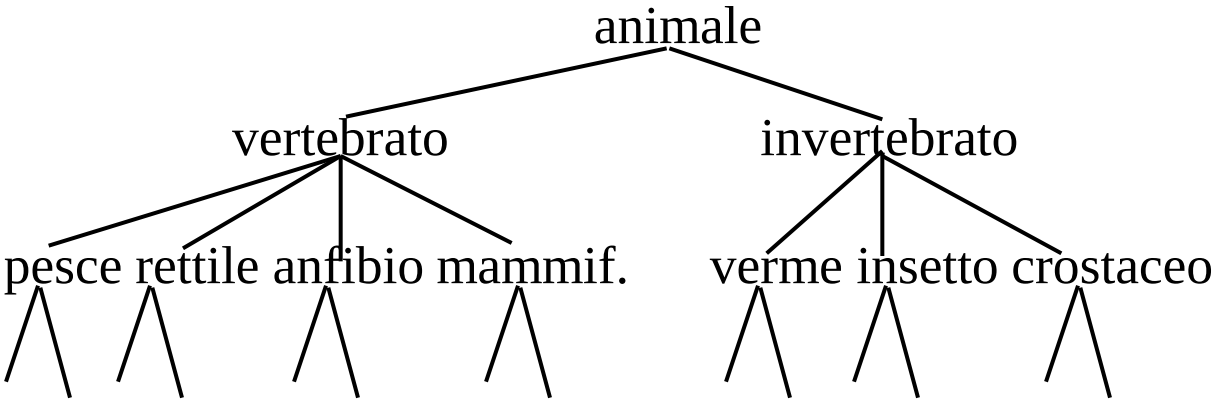
- Suddivide esempi non etichettati in sottoinsiemi disgiunti (**cluster**), tali che:
  - Gli esempi in uno stesso gruppo sono "molto" simili
  - Gli esempi in gruppi diversi sono "molto" differenti
- Scopre **nuove categorie** in modo **non supervisionato** (a priori non vengono fornite etichette per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

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

- 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

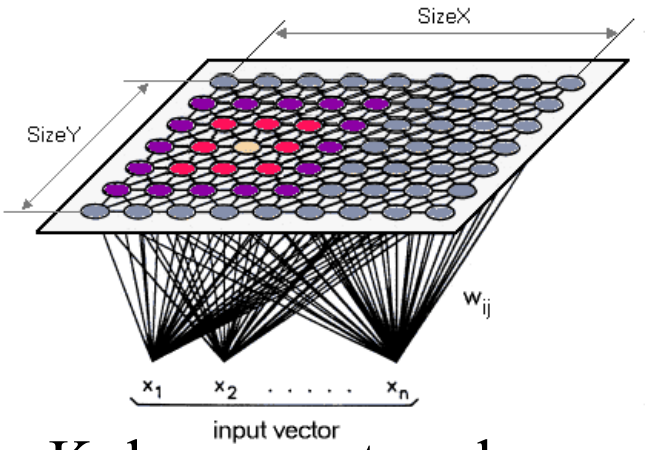
- Competitive Learning

  - Reti di Kohonen

- Reinforcement Learning

Apprendimento competitivo

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



Kohonen network

Si crea un isomorfismo tra spazio di input e di output

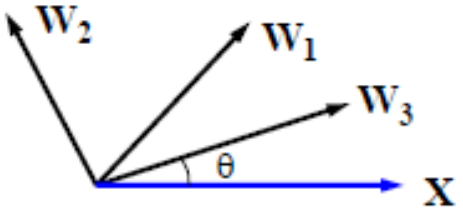
Implementazione

- 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;

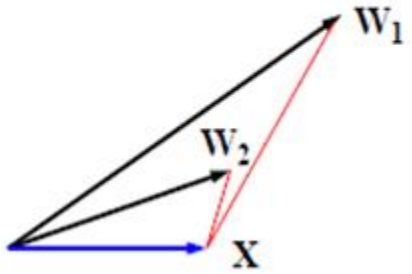
- 2. Seleziona il neurone il cui vettore peso è più simile all'input

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



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

$$j_0 : \mathrm{DIS}(W_{j0}, X) < \mathrm{DIS}(W_j, X) \quad \forall j \neq j_0$$



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

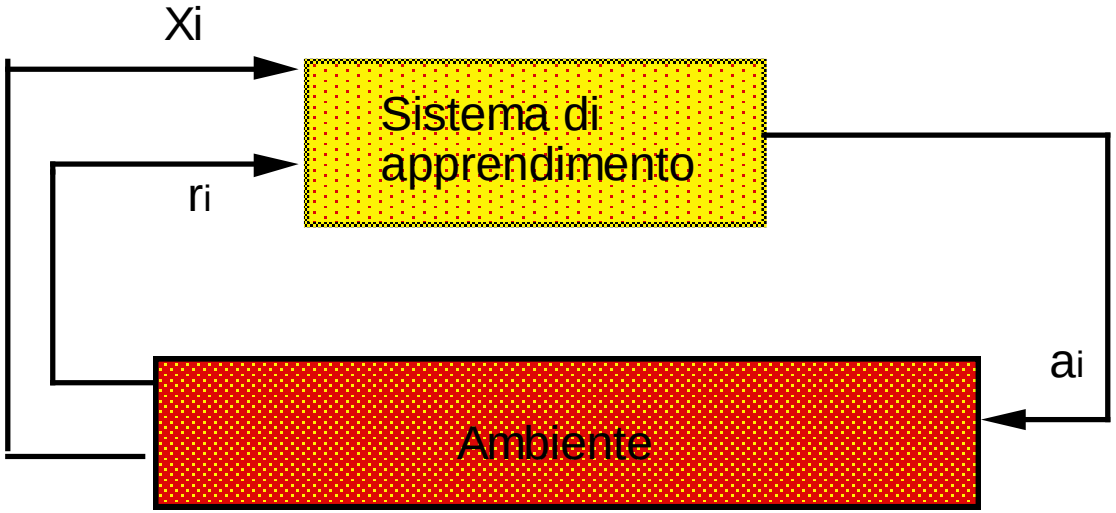
- Non sempre è possibile modellare un problema di apprendimento come la scelta ottimale di una funzione di classificazione
- L'apprendimento per rinforzo modella (tipicamente) il caso di un agente che percepisce ed agisce in un certo ambiente, il cui obiettivo è di imparare a fare "la cosa ottimale"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

Modello (2)

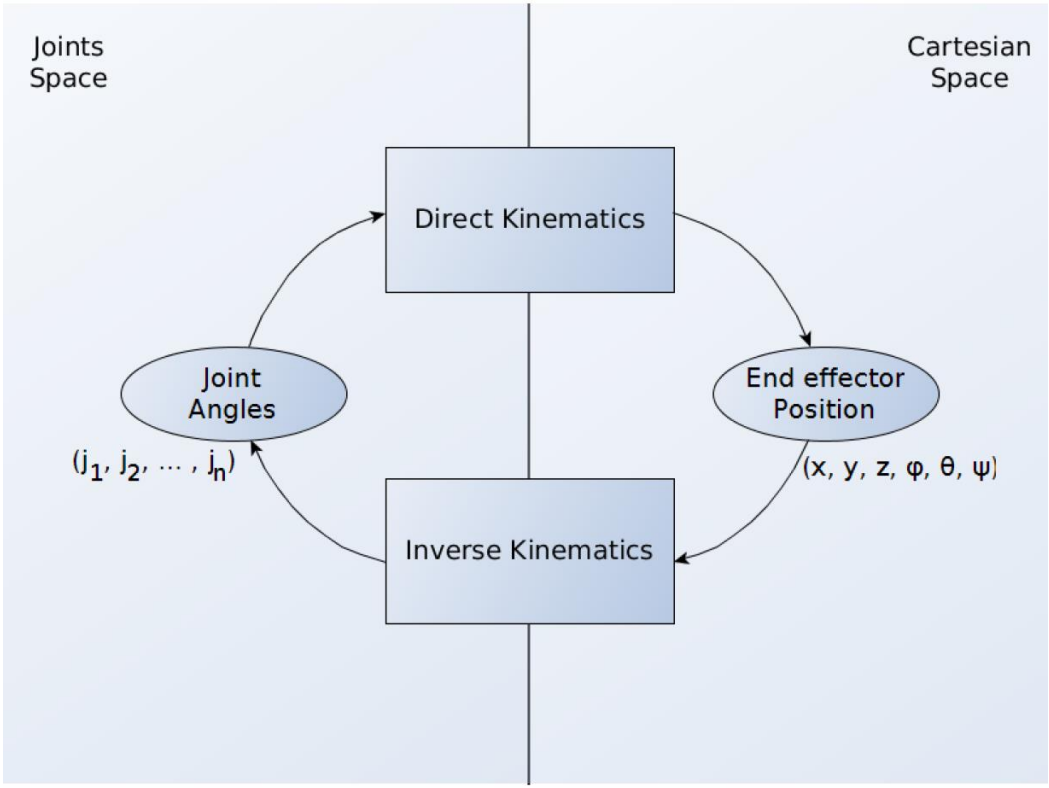
L'obiettivo di apprendimento per rinforzo consiste nell'identificare una politica ottima $\pi(X)$ $X \rightarrow$ A che massimizzi le ricompense accumulate nel tempo.

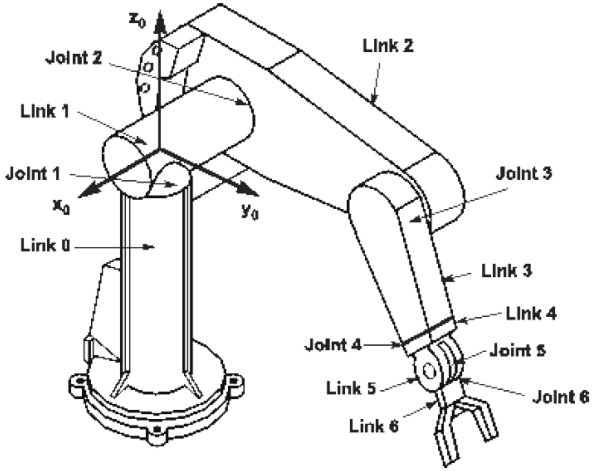


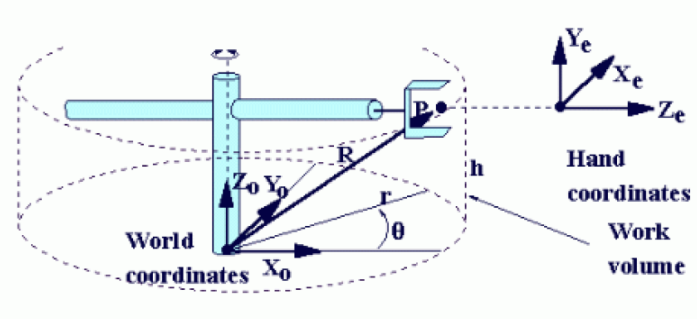
$$X_0 \xrightarrow{a_0} r_0 \quad X_1 \xrightarrow{a_1} r_1 \quad X_2 \xrightarrow{a_2} r_2 \ \ldots\ldots$$

Controllo robotico e predizione

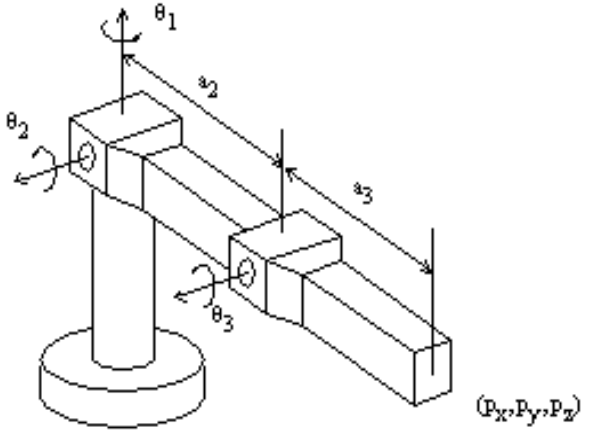
Controllo robotico



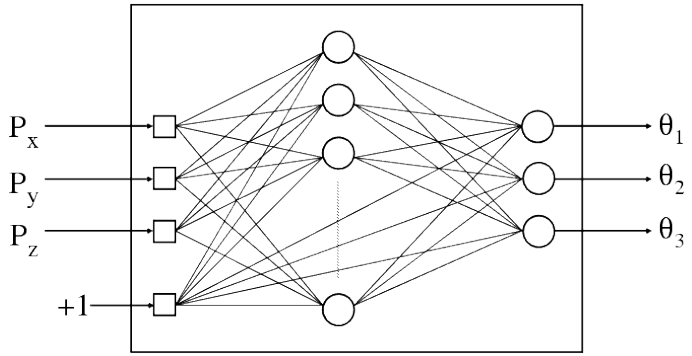




Calcolo Cinematica Inversa usando NN



How do you make the network learning?

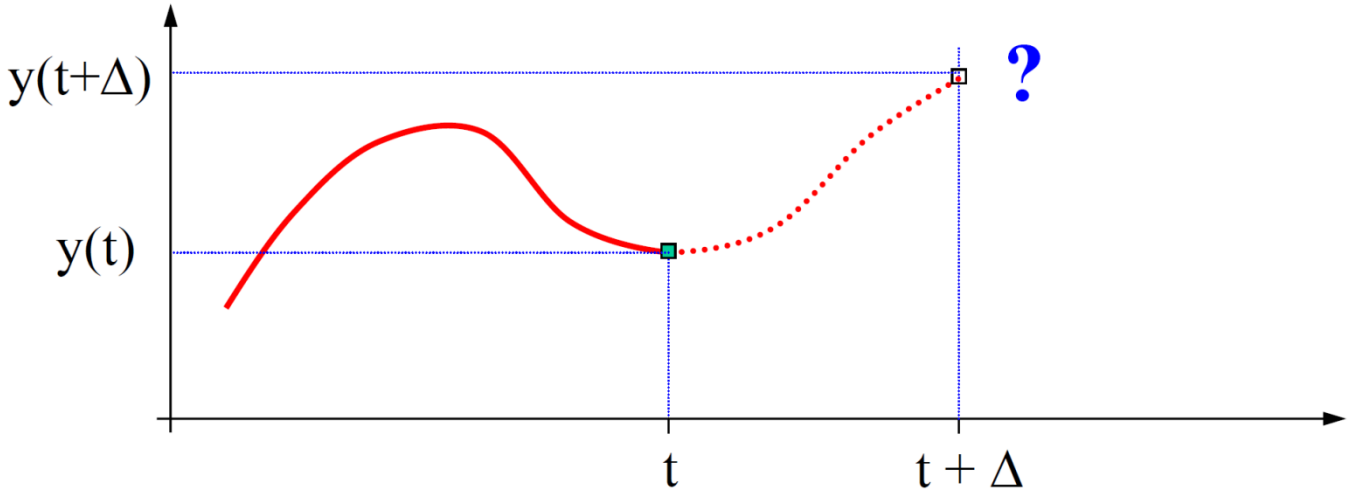


- ✓ Dataset di <posizione_giunto, posizione_end effector> tramite calcolo cinematica diretta

Predizione di segnali

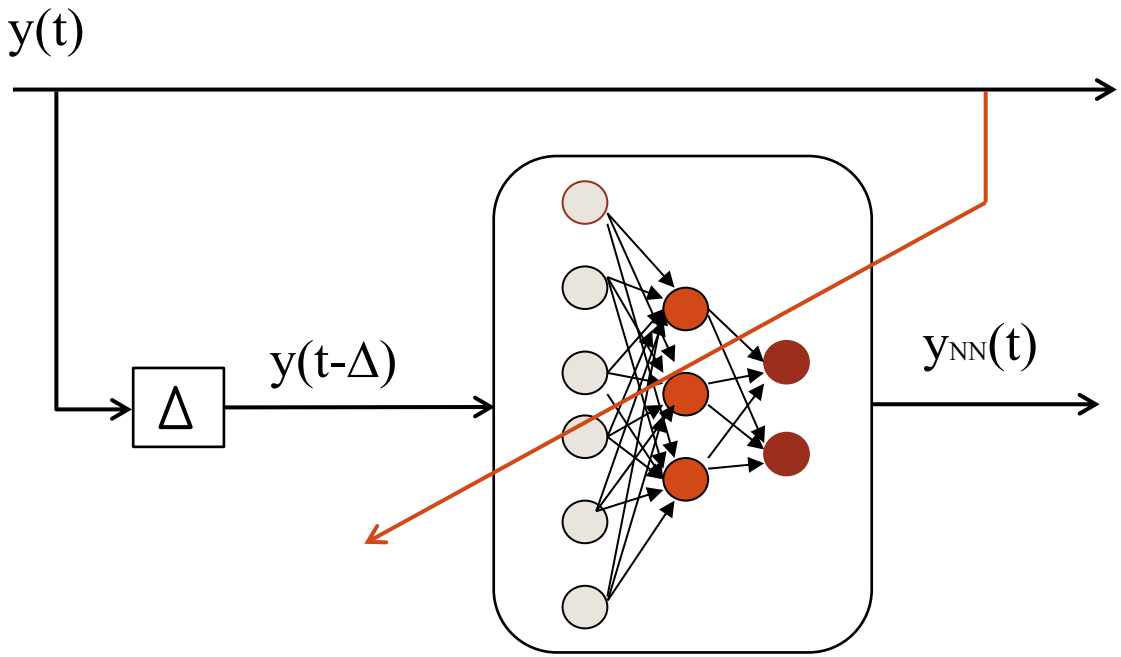
Usare Reti Neurali per predire il valore di un segnale nel futuro basandosi sulla storia segnale

Allenare la rete neurale utilizzando valori precedenti



Apprendimento e Predizione

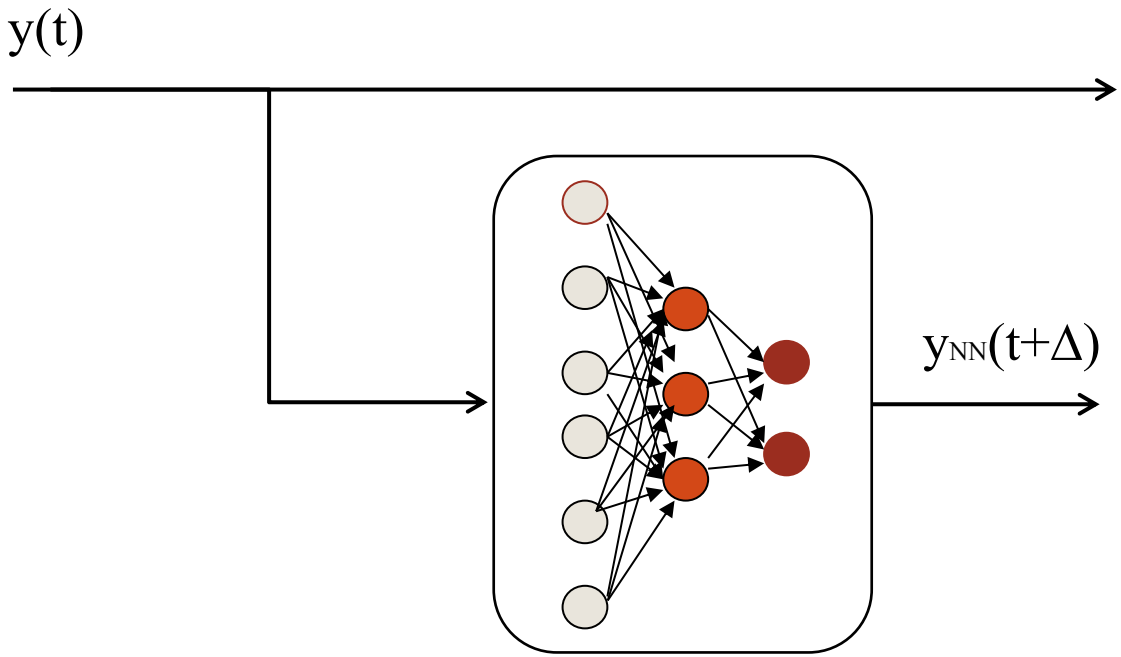
Fase di apprendimento



La rete neurale impara ad associare y(t) ad y(t-Δ)

Apprendimento e Predizione

Fase di test



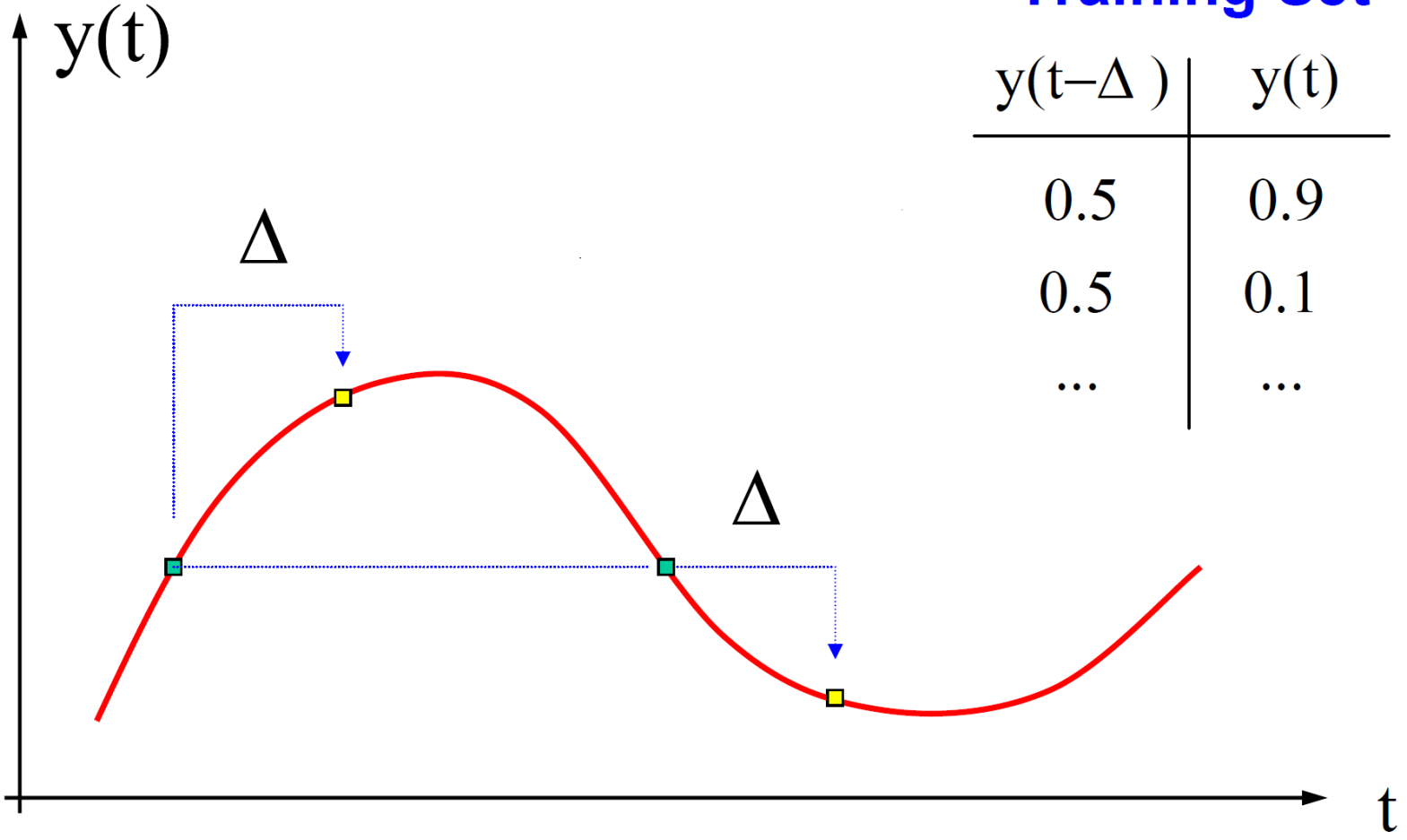
La rete neurale produce una stima di $y(t+\Delta)$

Apprendimento e predizione

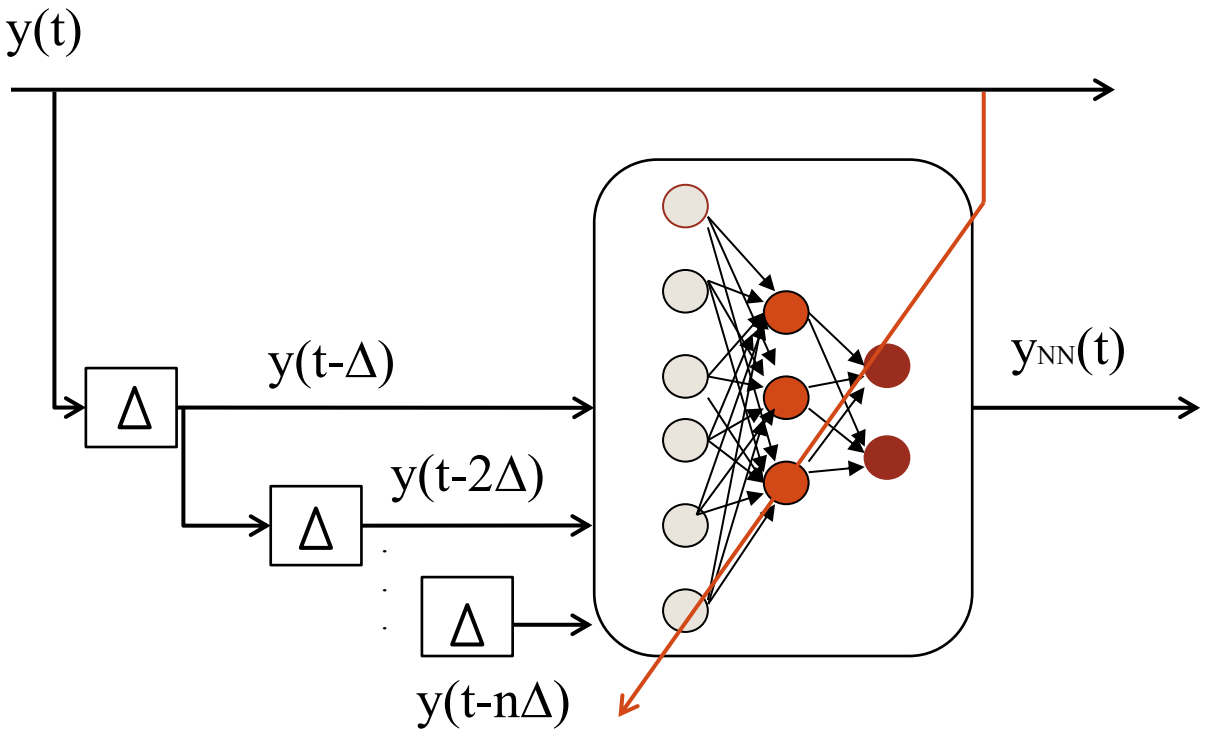
Training set non consistente

Training Set

$y(t-\Delta)$	$y(t)$
0.5	0.9
0.5	0.1
...	...

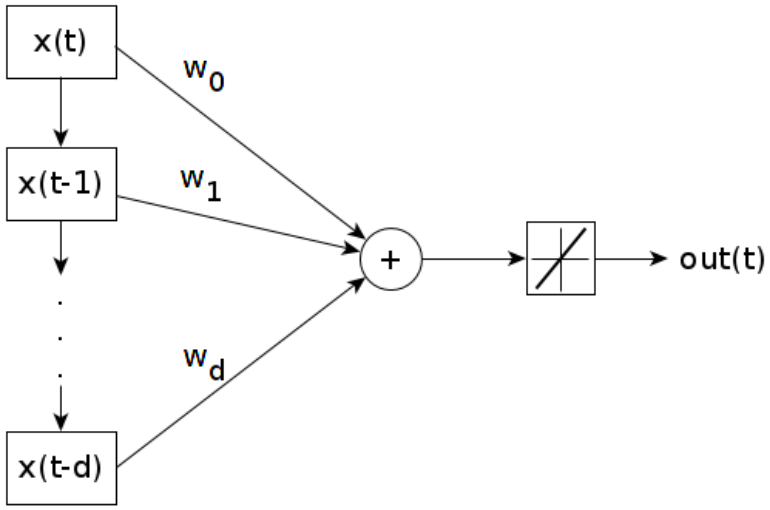


Soluzione: predizione con più ritardi



Predizione online

- Problema: effettuare la predizione senza alcuna conoscenza o precedente apprendimento sul segnale
- Rapido adattamento con pochi campioni a disposizione
- Modello semplice basato su un singolo neurone con ingresso i campioni correnti e passati (x(t)-x (t-d))



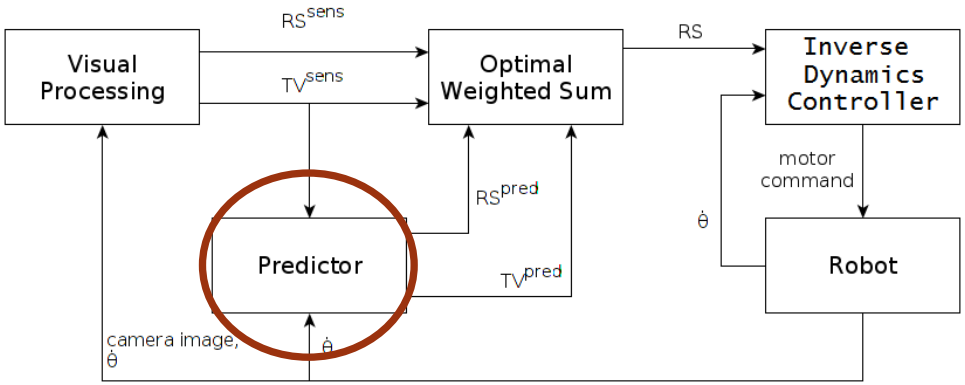
Es. predizione 10 passi in avanti con d=10

Tuple del training set :

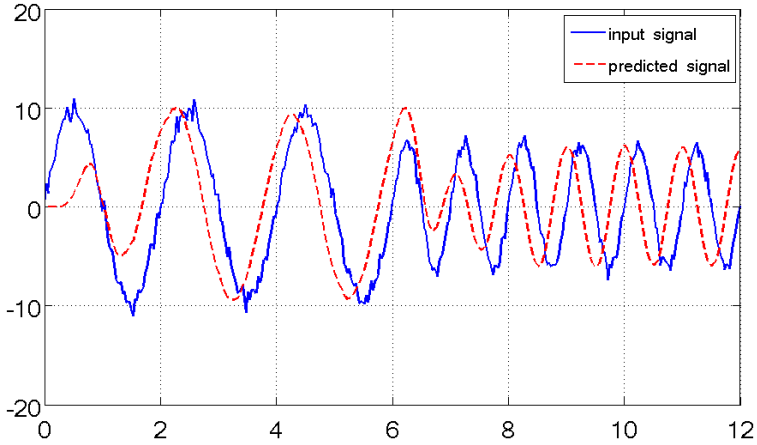
$\langle X_1\text{-}X_{10}, X_{20}\rangle, \langle X_2\text{-}X_{11}, X_{21}\rangle, \ldots \langle X_n\text{-}X_{n+10}, X_{n+20}\rangle$

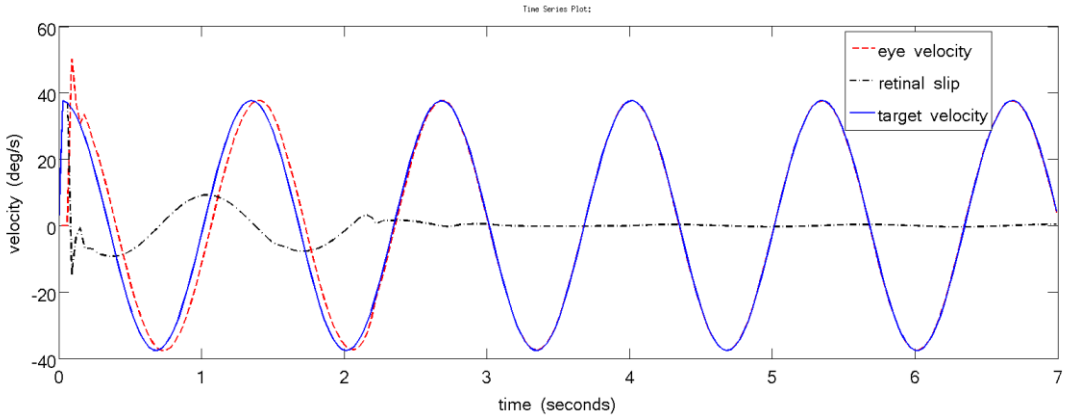
Uscite rete: $y_{10}, y_{11} \ldots y_{n+10}$

Esempio: inseguimento visivo basato su apprendimento online della dinamica del target



Predittore con segnale di ingresso rumoroso





Esecuzione task di inseguimento visivo

Esempi di neurocontrollori

1) 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

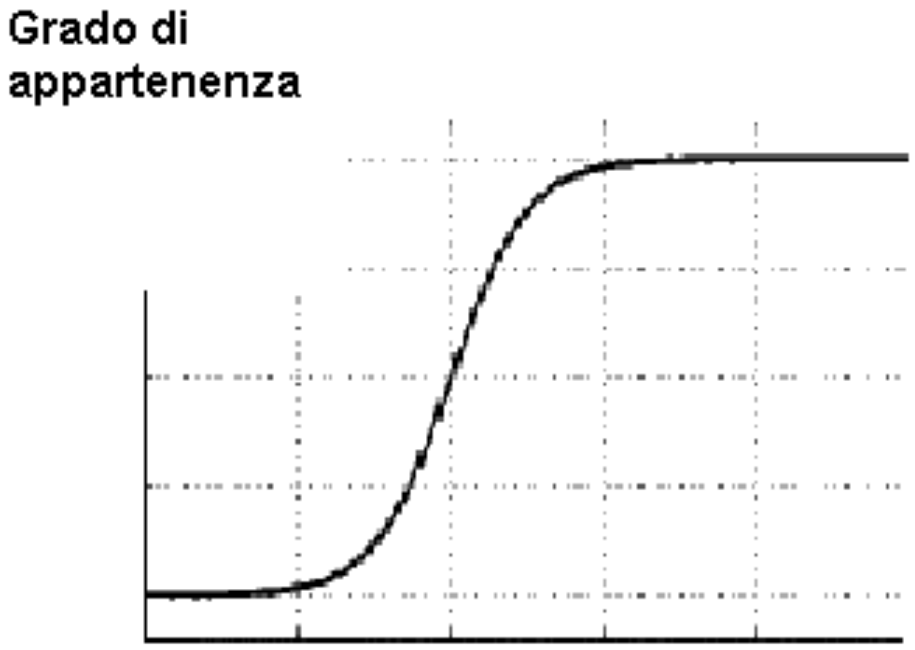
La funzione di appartenenza

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



Funzione di appartenenza della categoria fuzzy "ABBASTANZA ALTO"

Le regole della logica Fuzzy

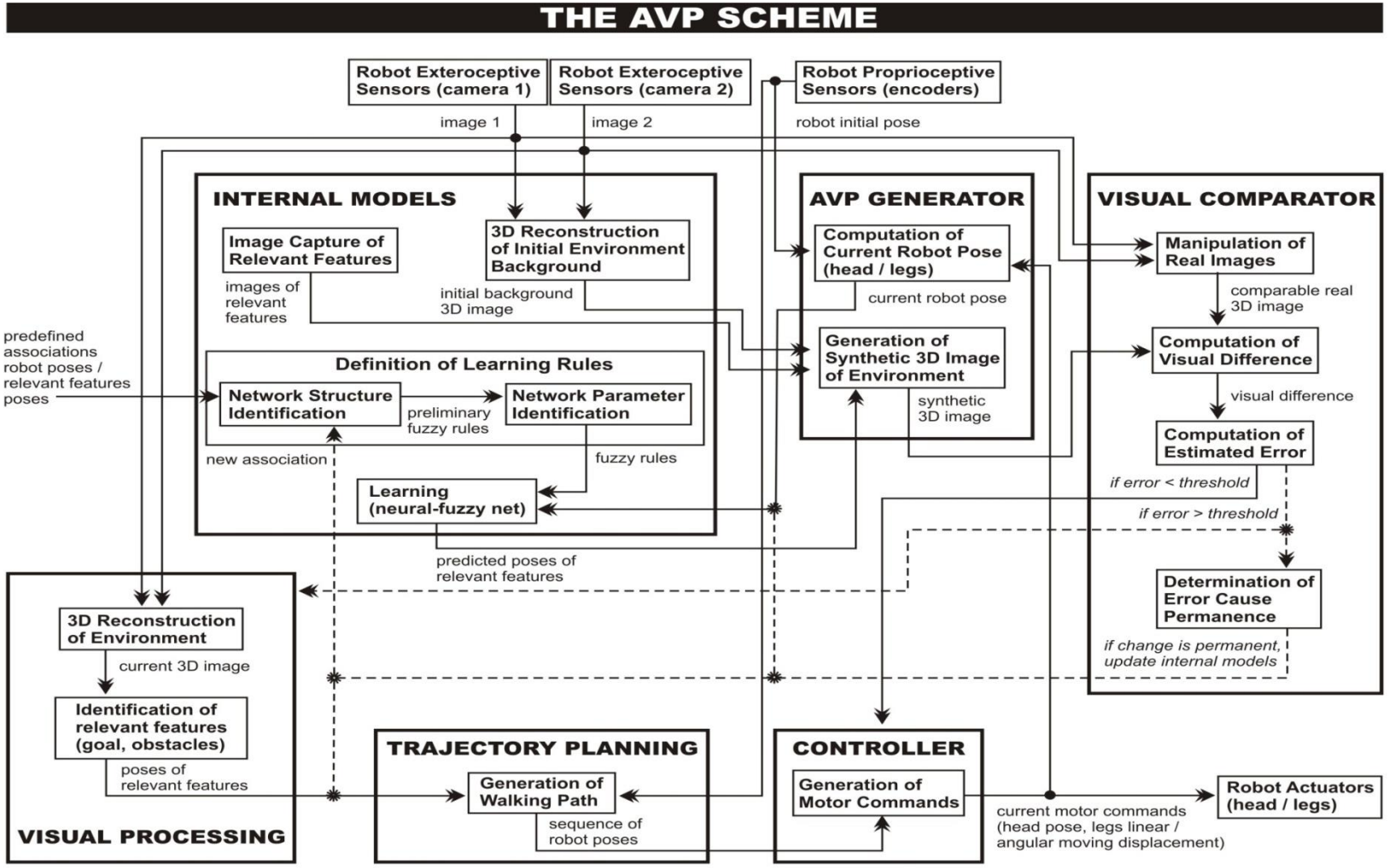
Usando i concetti fuzzy, si possono combinare tra loro con le regole della logica fuzzy per effettuare RAGIONAMENTI FUZZY

Ad esempio:

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

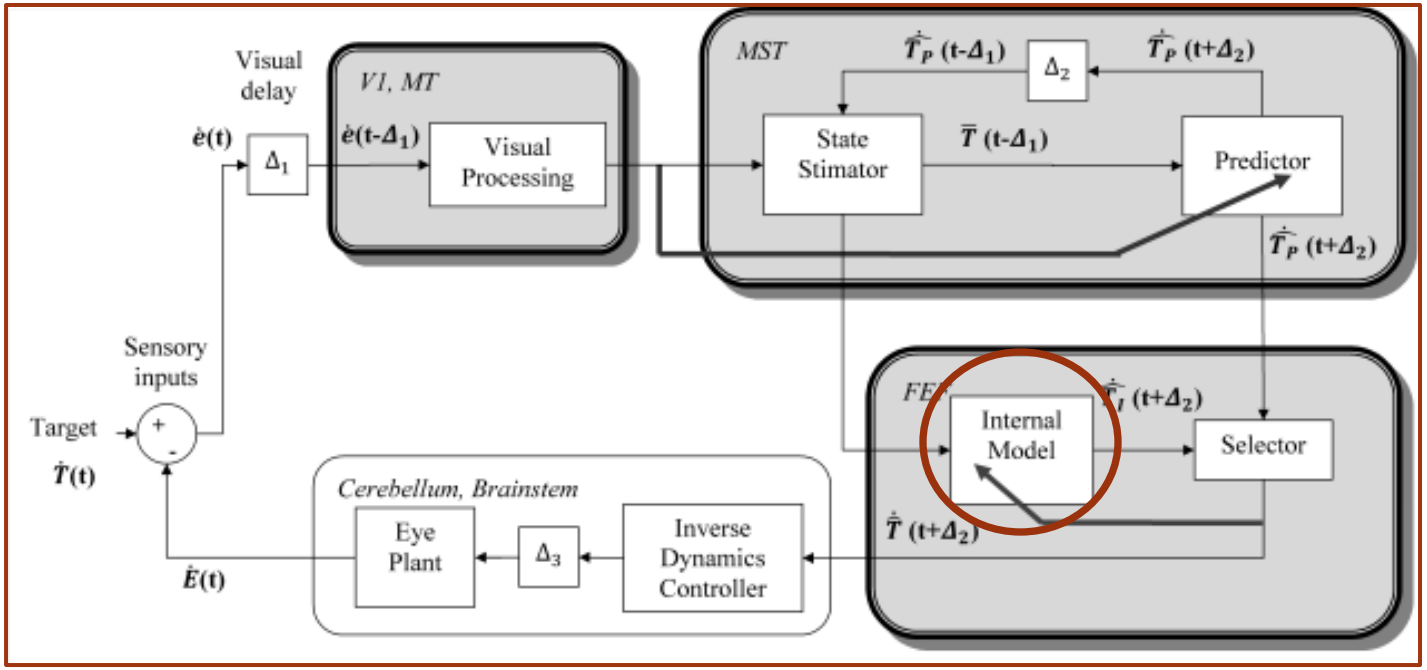
Il primo termine si chiama ANTECEDENTE e il secondo CONSEGUENTE

Lo schema AVP



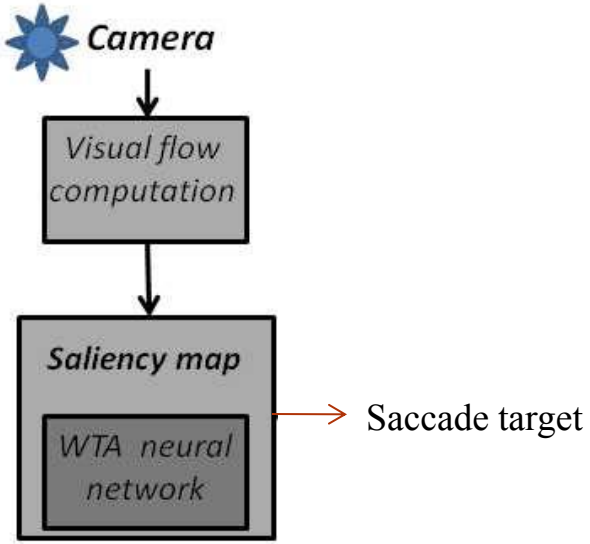
2) Rete neurale supervisionata per l'implementazione di modelli interni

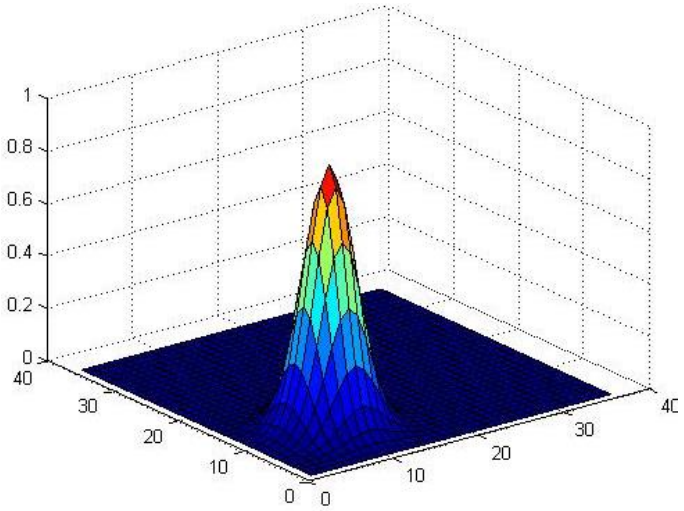
- Smooth pursuit



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

3) Competitive learning per la generazione di saliency map nella generazione di saccadi





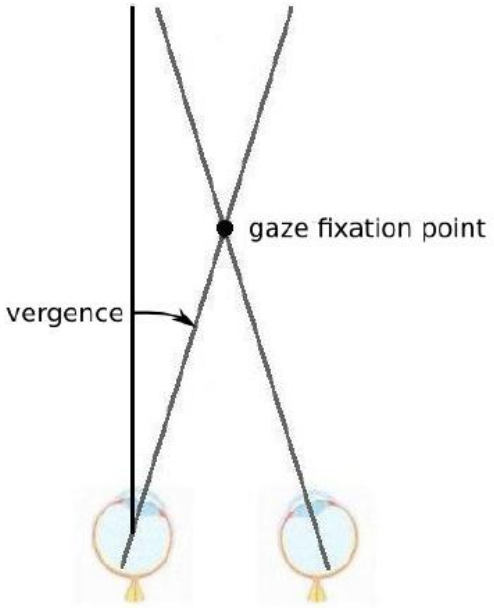
La rete neurale è utilizzata per modellare il processo di informazioni della corteccia celebrale (Amari et al,1977)

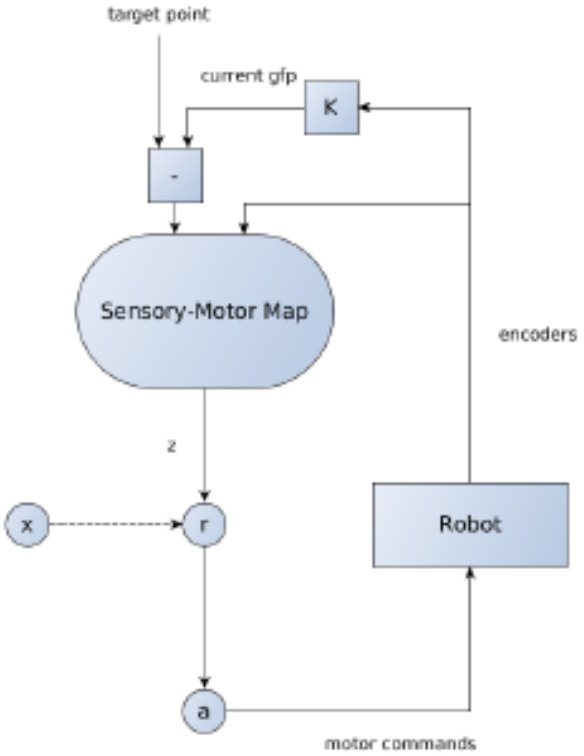
L'obiettivo è prendere in input stimoli spazialmente localizzati che competono per diventare il successivo punto target per il movimento oculare di saccadi

Modello di ompetitive learning per la generazione di mappe di saliency (video)

4) Un neurocontroller basato su Growing Neural Gas

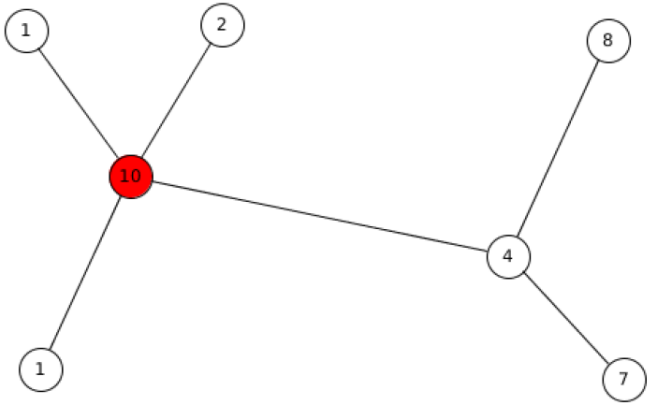
- ✓ Modello biologicamente ispirato
  - ✓ Mappa senso-motoria
  - ✓ Stimolo eccitatori/ inibitori



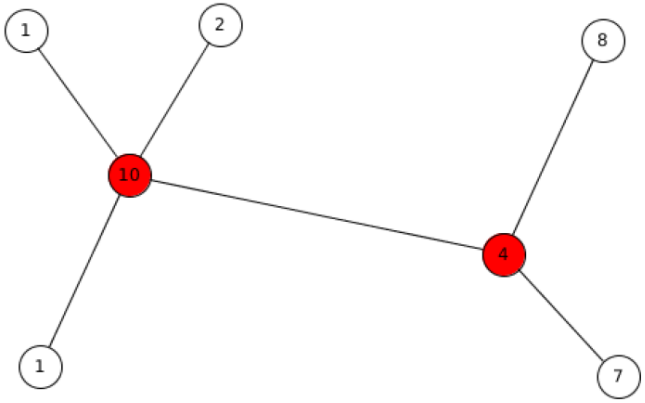


Asuni, G., Teti, G., Laschi, C., Guglielmelli, E., and Dario, P. (2005, April). A robotic head neuro-controller based on biologically-inspired neural models. In Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on (pp. 2362-2367). IEEE.

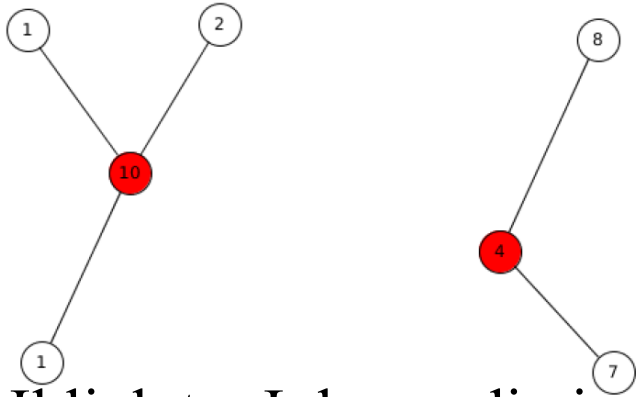
Modello di Growing Neural Gas



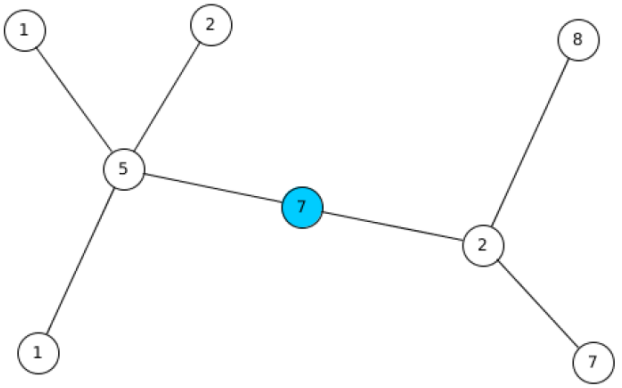
Il nodo con il più alto errore accumulato viene selezionato



Il vicino con l'errore maggiore è selezionato

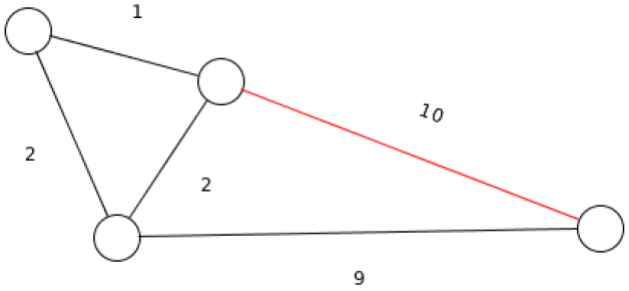


Il link tra I due nodi viene rimosso

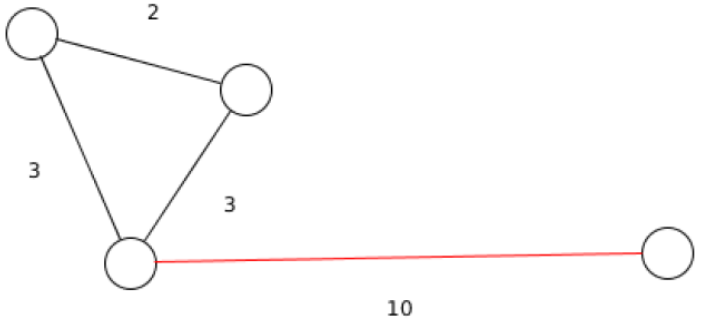


Una nuova unità viene aggiunta

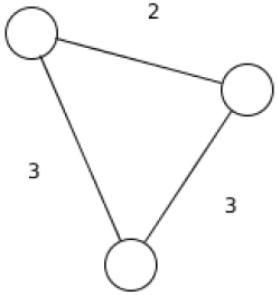
Growing Neural Gas model



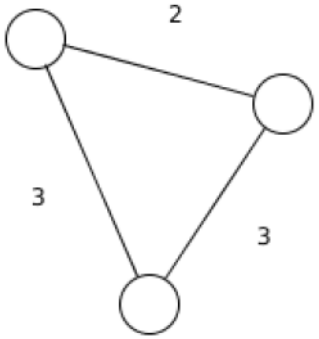
Tutti gli archi con età superiore a 10 sono marcati per la cancellazione



Di conseguenza un altro arco deve essere rimosso



L'unità non ha archi uscenti



L'unità è rimossa

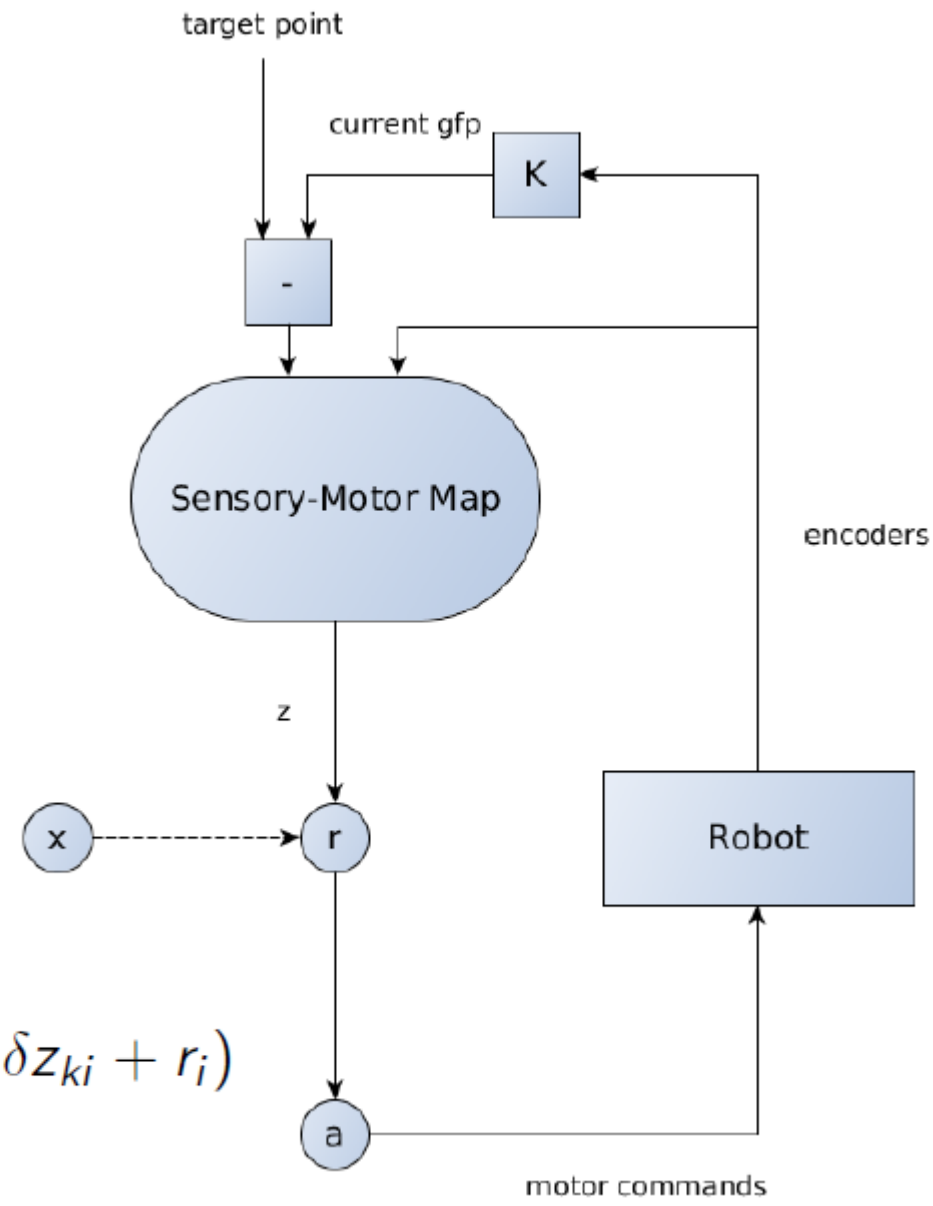
Apprendimento

L'apprendimento è realizzato in due fasi

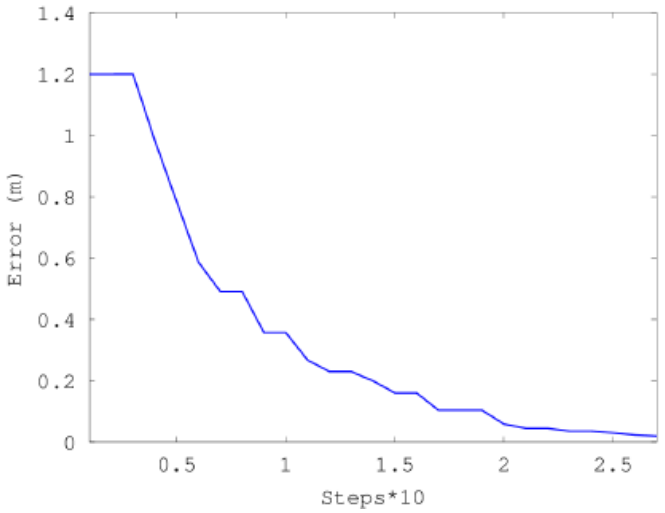
Fase 1: generazione della mappa senso-motoria, il training della GNG è effettuato indipendentemente

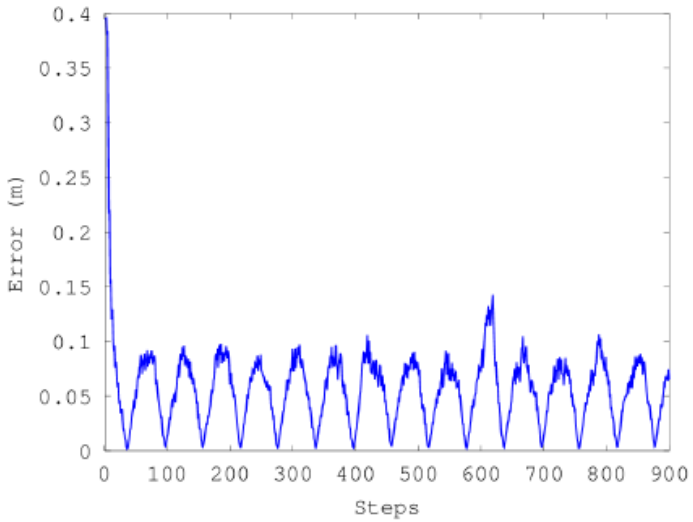
Phase 2: generazione di movimenti endogeni *x* (motor babbling) e aggiornamento dei pesi *z* a partire dalle unità vincitrici e dai vicini

$$\frac{\mathrm{d}z_{ki}}{\mathrm{d}t} = \gamma \cdot c_k \cdot (-\delta z_{ki} + r_i)$$



Risultati

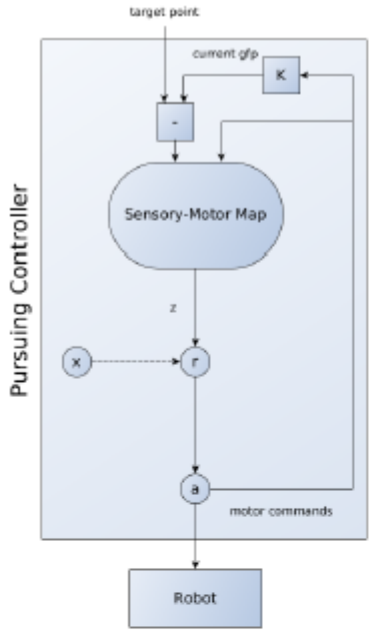


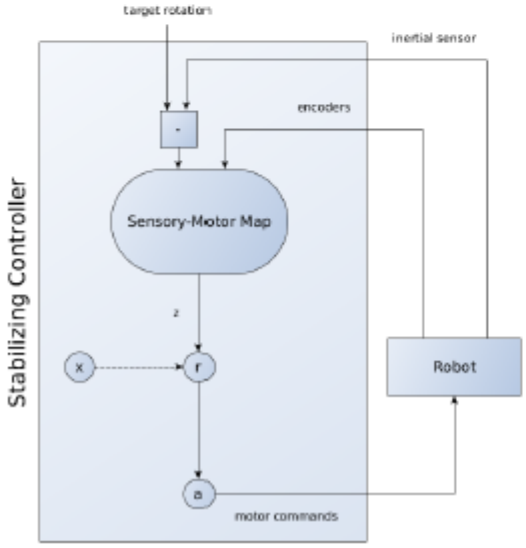


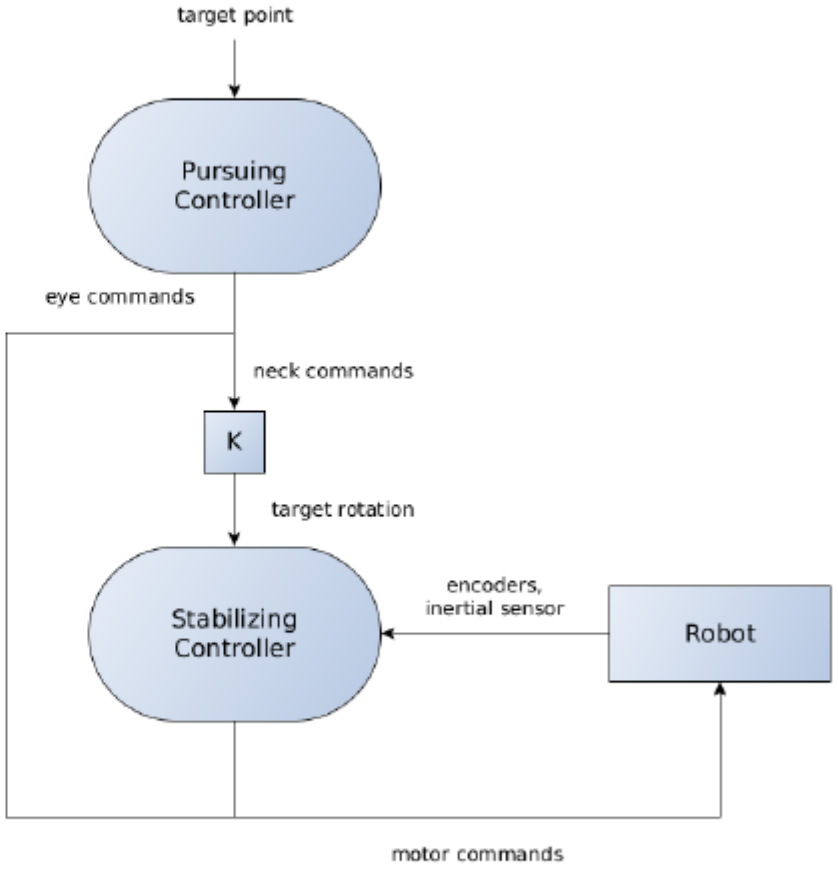
- ✓ Il controllore garantisce l'inseguimento di oggetti in movimento.
- ✓ Test effettuati con movimenti sinusoidali del target

f (Hz)	$\overline{E}$ (m)	E_{max} (m)	E_{min} (m)
0.1	0.022072	0.046741	0.000456
0.25	0.054736	0.142454	0.001733
0.5	0.092294	0.210824	0.002447
1	0.140470	0.274547	0.006962
2	0.160077	0.284362	0.008896

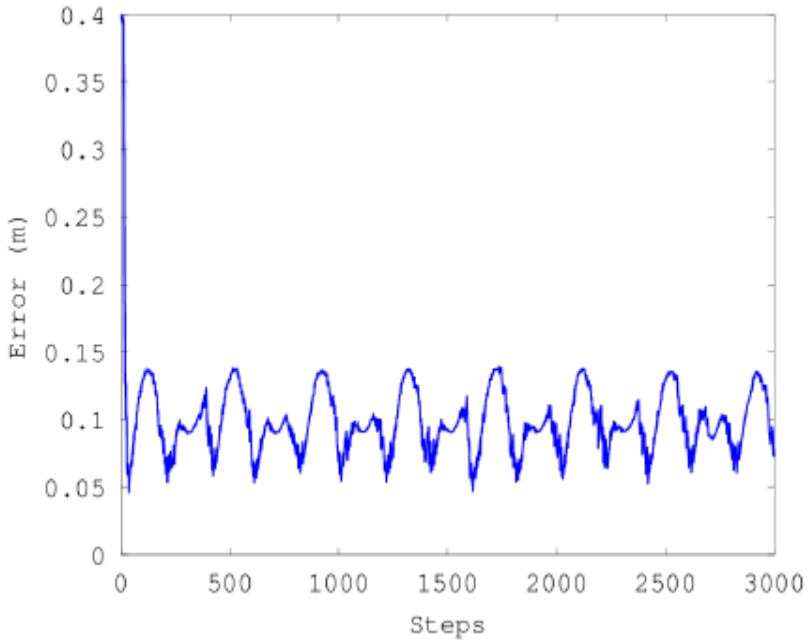
Controllore di inseguimento con stabilizzazione



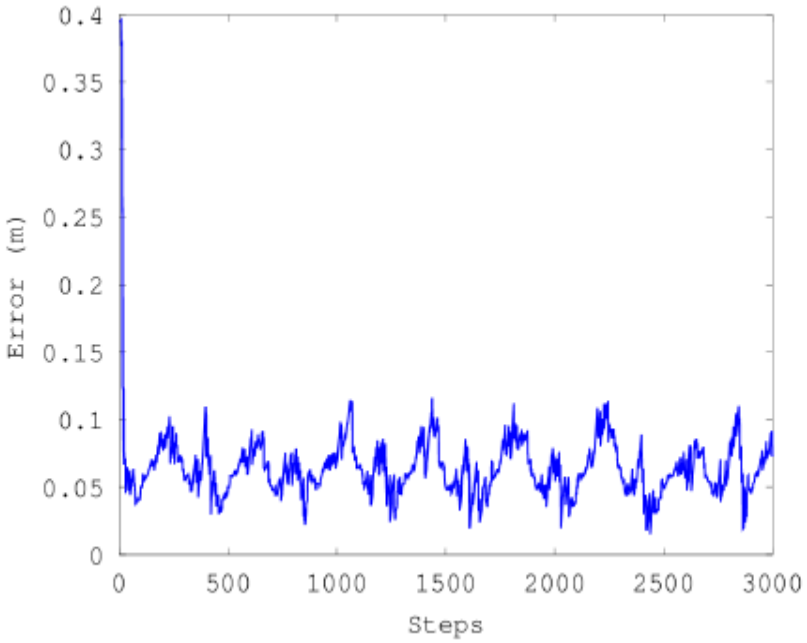




Risultati



Stabilizzazione attiva:
E=0.0983 m

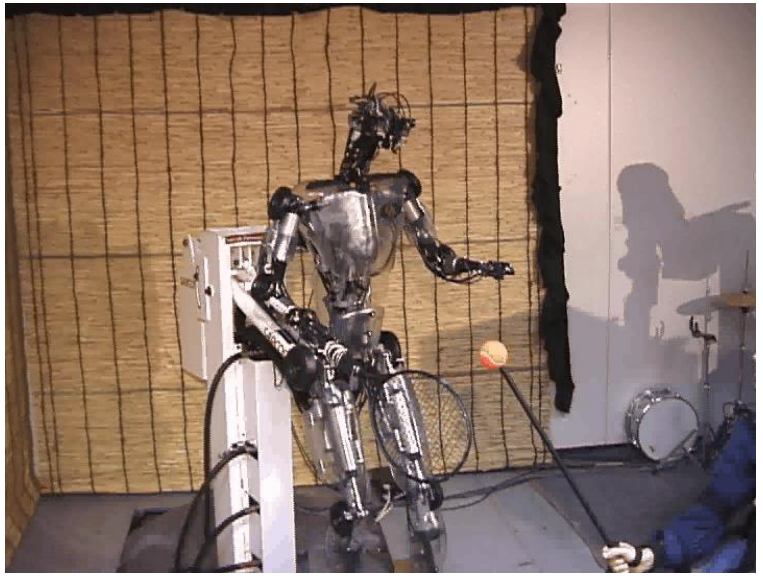


Stabilizzazione disattivata:
E=0.063 m

5) Un Passo avanti: Learning from demostration

- 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 demostration (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.