Reti Neurali

Apprendimento di funzioni algebriche

Argomenti

- Storia
- Funzionamento di un neurone
- Il percettrone
- Reti feed-forward
- L'algoritmo di apprendimento
- Esempi
- Applicazioni
  - Identificazione e controllo con reti neurali

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.

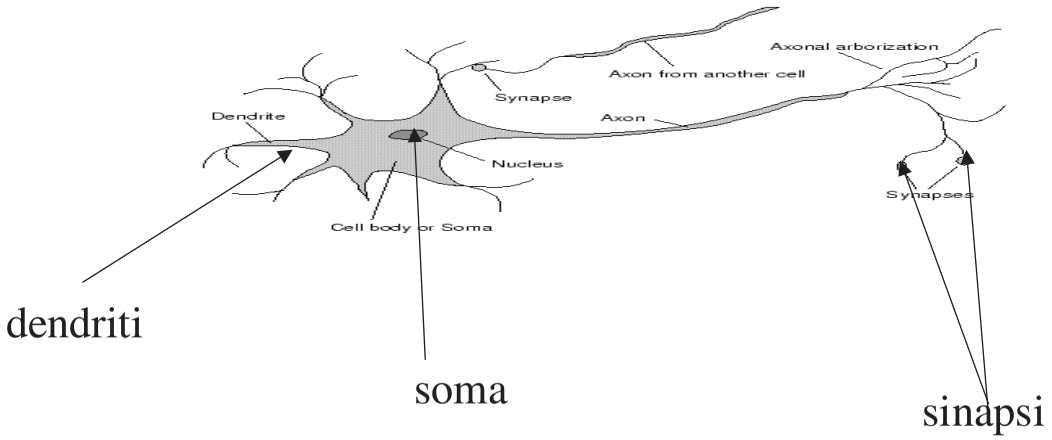
Due categorie di modelli

- 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 sinapisi 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

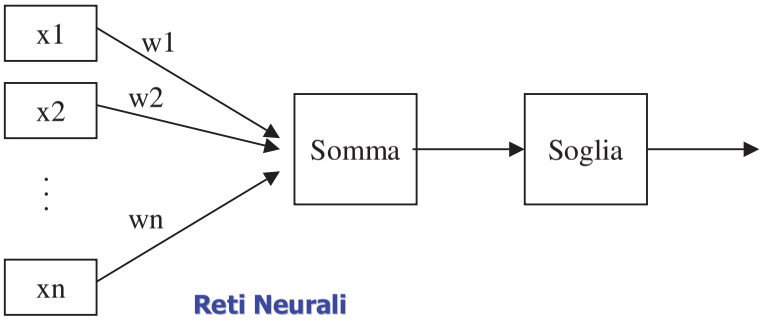
Caratteristiche:

- 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

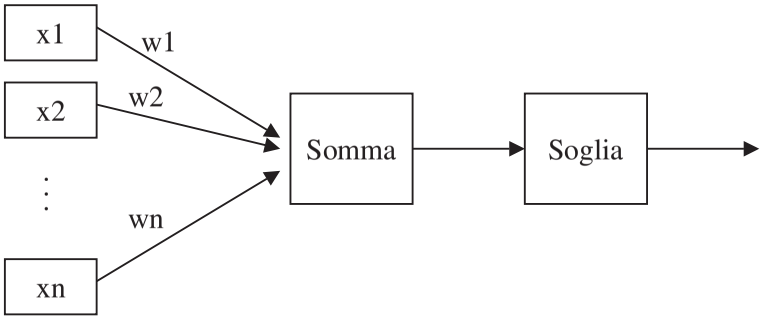
Il Percettrone

- 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



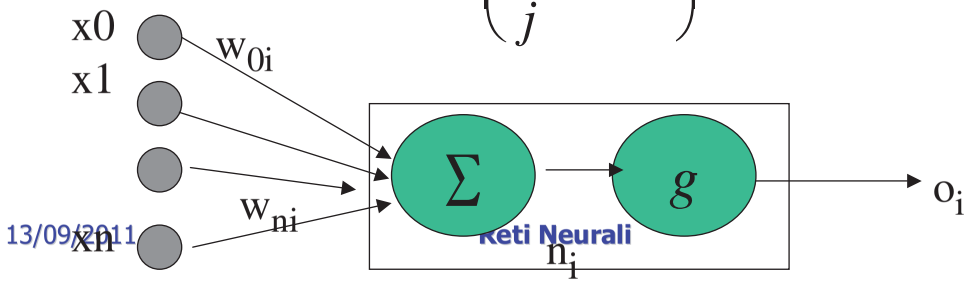
Funzioni Somma e Soglia

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

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

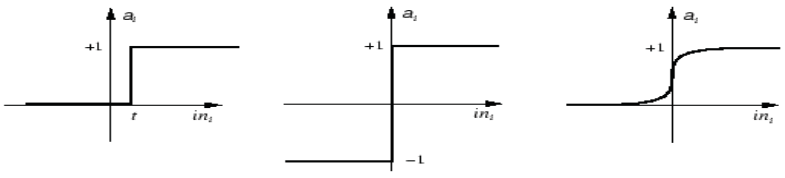
b) **funzione di attivazione**, *non lineare (SOGLIA)*

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



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 forma vettoriale mediante la:

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

Il Percettrone

- 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

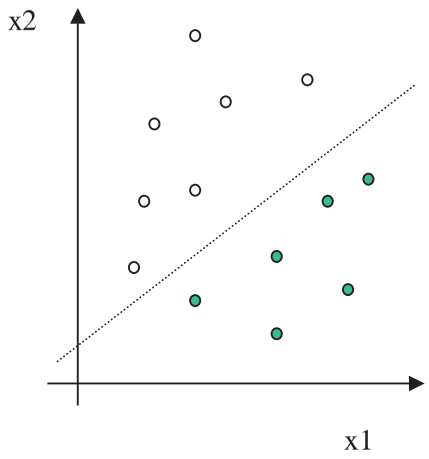
$$H = \left\{ \vec{w} : \ \vec{w} \in \Re^{n+1} \right\}$$

- 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
- 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 $\vec{w} \cdot \vec{x}$
- Se per esempio: $x_i = 0,8, \ \eta = 0,1, \ c = 1, \ o = -1$

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

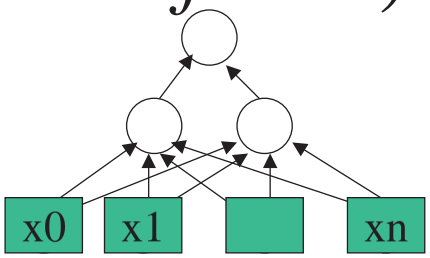
- Quindi il valore dei w_i tenderà a crescere, riducendo l'errore.
- Se invece c(x)=-1 e o(x)=1

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

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**

1. Reti Alimentate in Avanti (*stratificate*)



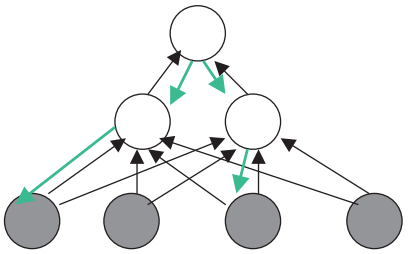
- Ogni unità è collegata solo a quella dello strato successivo.
- L'elaborazione procede uniformemente dalle unità d'ingresso a quelle di uscita
- Non c'è feedback (grafo aciclico direttoo DAG)
- Non ha stato interno

2. Reti 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.



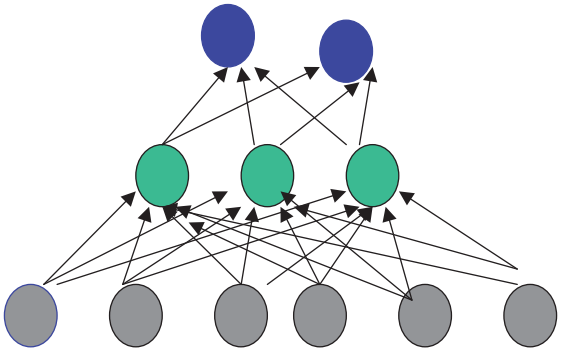
Esempi

- Macchine di Boltzmann
- Reti di Hopfield

Reti 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

Back-Propagation (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:

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

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

t(x)=valore del concetto in x∈D

Back-Propagation (3)

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

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

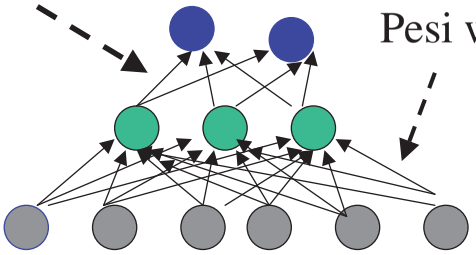
$$\Delta w_{ji} = \eta \delta_j x_{ji}$$

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

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

Pesi w_{ij} (da n_j a n_i)

Pesi w_{ji}



- Ii Unità di ingresso
- Hj Unità nascoste
- Oi Unità di uscita

Algoritmo Back-Propagation (4)

- D è un insieme di coppie (x,t(x)), dove x è il vettore dei valori di ingresso (x_1,x_2..) e t è il vettore dei valori t_1,..t_m della funzione obiettivo t
- η 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* fra -0,05 e 0,05

Algoritmo Backpropagation (5)

- **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: $w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$

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

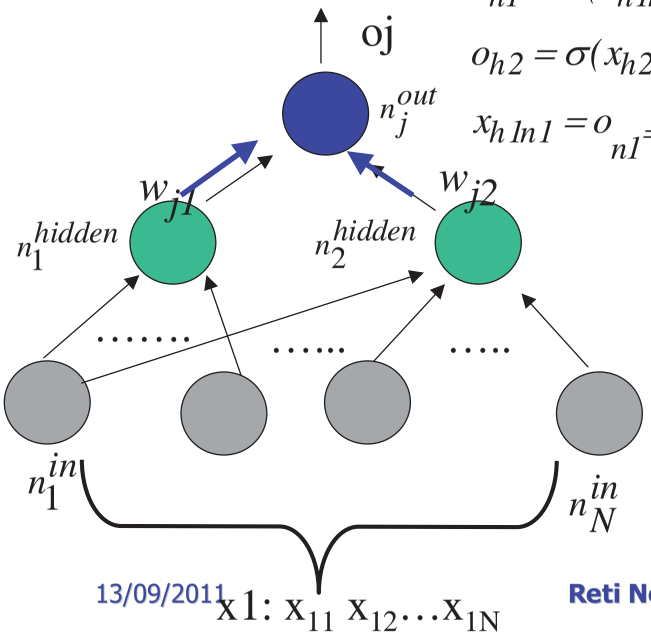
Esempio

$$o_j = \sigma(x_{njh1}\ w_{j1} + x_{njh2}\ w_{j2}\ ) = \sigma(o_{h1} w_{j1} + o_{h2}\ w_{j2}\ )$$

$$o_{h1} = \sigma(x_{h1n1} w_{n1h1} + x_{h1n2} w_{n2h1} + ... + x_{h1n_N} w_{n_N h1})$$

$$o_{h2} = \sigma(x_{h2n1} w_{n1h2} + x_{h2n2} w_{n2h2} + ... + x_{h2n_N} w_{n_N h2})$$

$$x_{h1n1} = o_{n1=\sigma(x11)} ... o_{nN} = \sigma(x_{1N})$$



$$x_{ji} = o_{ji}$$

Passo 1: si alimenta col primo esempio e si calcolano tutte le oj

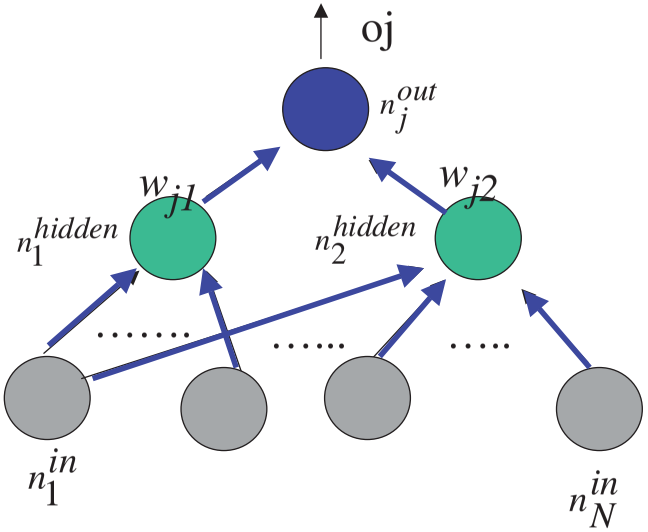
Esempio (continua)

$$\delta_j = o_j(1 - o_j)(t_1 - o_j)$$

$$\delta_{h1} = x_{h1}(1 - x_{h1}) w_{j1} \delta_j =$$

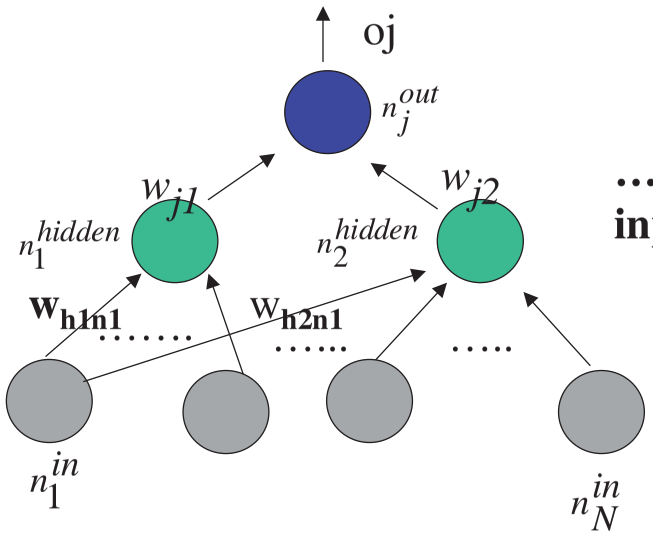
$$o_{h1}(1 - o_{h1}) w_{j1} \delta_j$$

$$\delta_{h2} = o_{h2}(1 - o_{h2}) w_{j2} \delta_j$$



Si calcola l'errore sul nodo di uscita e, all'indietro, sui nodi hidden

Esempio (3)



…e sui nodi di input

$$\delta(n_1^{in}) = o_1^{in}(1 - o_1^{in})(w_{h1n1}\delta_{h1} + w_{h2n1}\delta_{h2})$$

.....

$$\delta(n_N^{in}) = o_N^{in}(1 - o_N^{in})(w_{h1n_N}\delta_{h1} + w_{h2n_N}\delta_{h2})$$

Esempio(4)

$$w_{j1} \leftarrow w_{j1} + \eta\delta_j x_{j1}$$

$$w_{j2} \leftarrow w_{j2} + \eta\delta_j x_{j2}$$

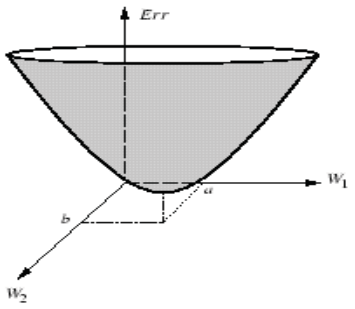
$$w_{h1n1} \leftarrow w_{h1n1} + \eta\delta_{h1} x_{h1n1} .....$$

- Si aggiornano tutti pesi
- Si considera il secondo esempio di D
- Si ricalcolano input e output di tutti i nodi
- Si ricalcolano gli errori sui nodi
- Si riaggiornano i pesi
- Finché non si esaurisce D (Epoca)
- Si ripete per n epoche, fino a che l'errore non si stabilizza

Spiegazione della Regola di Propagazione dell'Errore all'Indietro

Consideriamo la funzione di errore per x∈D:

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



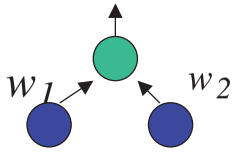
Supponiamo di dover imparare solo due pesi.
Il piano w1 w2 rappresenta lo spazio delle ipotesi (pesi delle connessioni) , il cono è la superficie d'errore.
Per minimizzare l'errore si deve calcolare la direzione della discesa più ripida lungo la superficie.
Quindi, la derivata.

Esempio di Calcolo del Gradiente

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

$$net_1 = w_1 x_1 + w_2 x_2$$

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



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

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

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

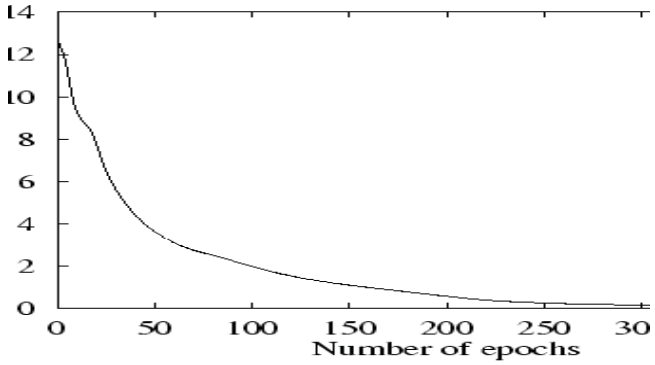
$$\Delta w_1 = \eta o(1 - o)(t - o)x_1$$

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'∪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 decisamente peggiori rispetto al minimo globale
  - Possono esserci molti minimi locali
- Soluzioni possibili: addestra la rete con pesi iniziali diversi, addestra diverse architetture di rete

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!!

Conclusioni

- 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!

Un Esempio

Riconoscimento di fisionomie (face recognition)

http://www2.cs.cmu.edu/afs/cs.cmu.edu/user/avrim/www/ML94/face_homework.html

Il compito:

Classificare immagini video di visi di persone in varie pose.

Data Set:

Immagini di 20 persone in (circa) 32 pose, variabili per:
espressione,
direzione dello sguardo, occhiali da sole (si/no), sfondo

Funzioni "target""Riconoscere" chi porta occhiali, in che direzione guarda, quale persona é..

Scelte di Progetto

Accuratezza della codifica di ingresso:

Una scelta chiave consiste nel decidere quali e quante "features" codificare.
Ad esempio, si potrebbe pre-analizzare l'immagine estraendo contorni, regioni di intensità luminosa uniforme, ecc.

Problema: il numero delle features sarebbe variabile (ad esempio, i contorni) mentre una NN ha un numero fisso di unità di ingresso.
➔ codifica l'immagine come un insieme di valori di intensità di pixels 30x32 (un input per pixel)

Codifica di uscita:
Dipende dalla funzione. Ad esempio, vogliamo sapere se la persona guarda a sinistra, destra, dritto, o in alto.
Possiamo usare 4 uscite (1-out-of-n output encoding) oppure definire una sola uscita con 4 ranges di valori.
La prima soluzione dà più gradi di libertà alla rete (ci sono più "pesi" da determinare). Inoltre, la differenza fra i valori degli output è una misura della "confidenza".

Struttura della Rete

- L'algoritmo di Backpropagation può essere applicato a qualsiasi grafo aciclico diretto, con funzione di attivazione sigmoidale.
- La struttura più comune è stratificata, con un numero di strati interni da uno a tre (>3 i tempi di training diventano inaccettabili), in cui ogni unità dello strato i è collegata con ogni unità dello strato i+1.

Quanti strati interni, quante unità in ogni strato?

Esperimento: con solo 3 unità nascoste e un solo strato interno, 90% precisione. Con 30 unità, solo +2%
Con 260 immagini per il training, 1 ora di tempo di addestramento su una Sun Sparc (30 unità) e solo 5 minuti con 3 unità.

Questo è un risultato abbastanza generale: un numero piccolo di unità nascoste consente di ottenere buoni risultati. Piccoli miglioramenti si ottengono a fronte di grosse variazioni dei parametri, causando, se non si prendono gli accorgimenti necessari, tendenza a overfitting.

Altri Parametri di Apprendimento

- η (tasso di apprendimento)=0,3
- α (momentum) = 0,3
- I pesi iniziali sulle unità di uscita sono *random*
- Le unità di ingresso sono inizializzate con un peso 0, perché ciò porta ad una migliore visualizzazione dei pesi appresi.

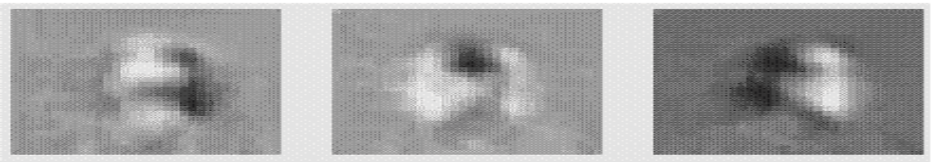
Overfitting, Generalizzazione, Criteri di Stop

Un problema importante è capire quando arrestare il processo di terazione (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 traing termina quando i risultati cominciano a divergere.

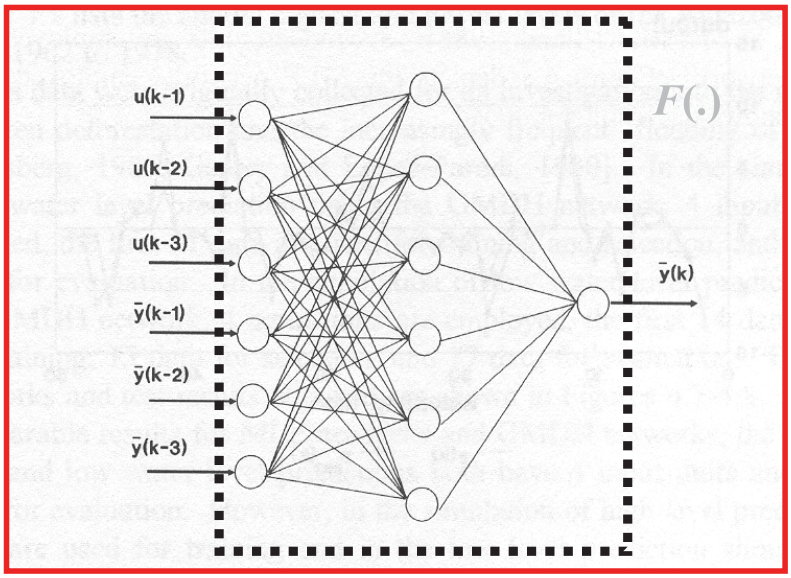
Dopo Alcune Iterazioni...

Immagini apprese dopo 100 iterazioni



Sistemi Dinamici Non Lineari

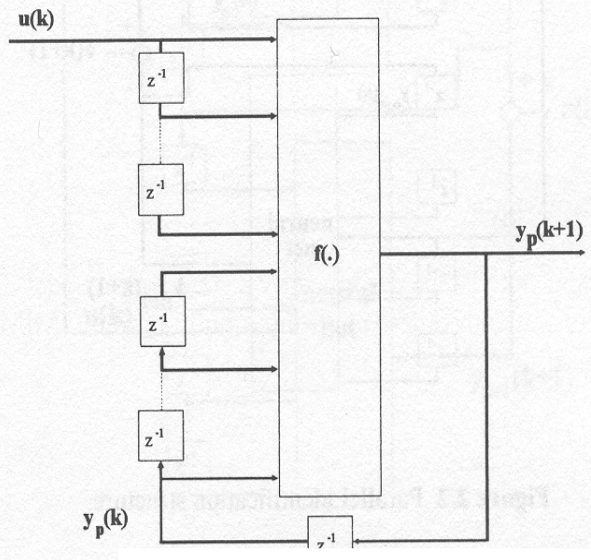
- Rete neurale statica
- Come passare dalle reti neurali statiche a quelle dinamiche?
- L'approccio più semplice è quello di considerare le reti "quasi-statiche"
- Si aggiungono ingressi, uscite segnali ritardati



$$\tilde{y}(k) = F\left(u(k-1), u(k-2), u(k-3), \tilde{y}(k-1), \tilde{y}(k-2), \tilde{y}(k-3)\right)$$

Esempio di rete neurale quasi-statica che 3 ingressi e uscite ritardati di 3 passi

Sistemi Dinamici Non Lineari Identificazione



Modello ingresso-uscita

Figure 2.1 Input-output model

- f(.), funzione incognita obiettivo
- Modello non lineare dinamico
- Approssimazione mediante una rete neurale quasi-statica
- **Identificazione di sistemi dinamici non lineari**
- Confronta con l'identificazione di sistemi dinamici lineari

Identificazione di Sistemi Dinamici Non Lineari

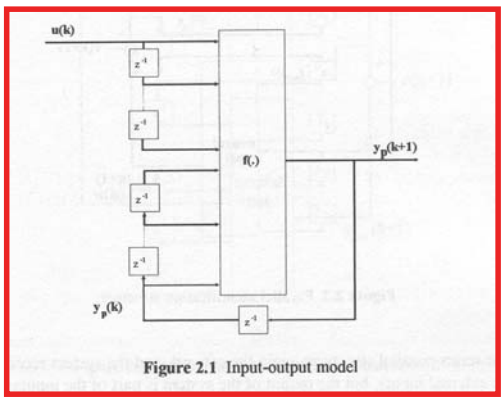
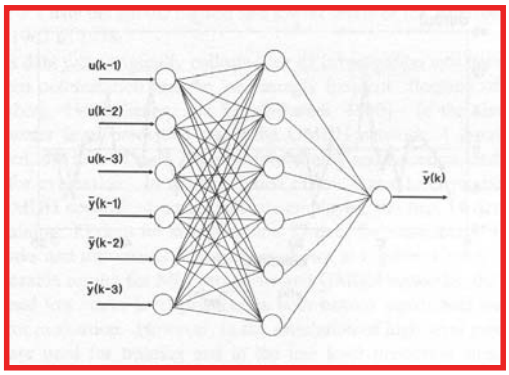
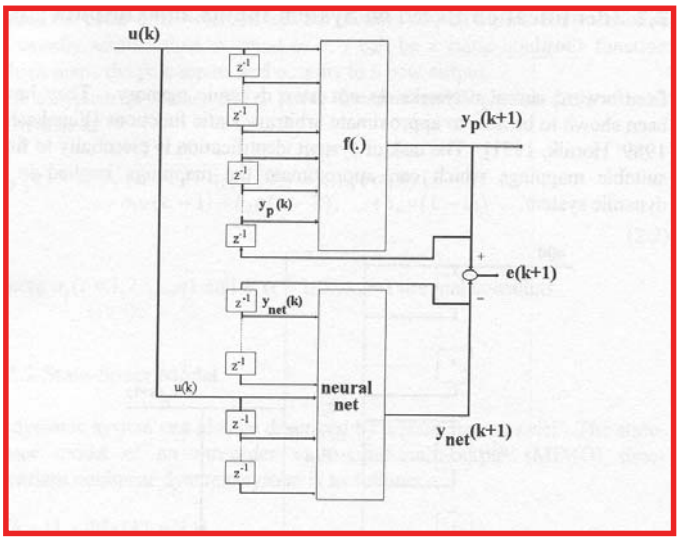


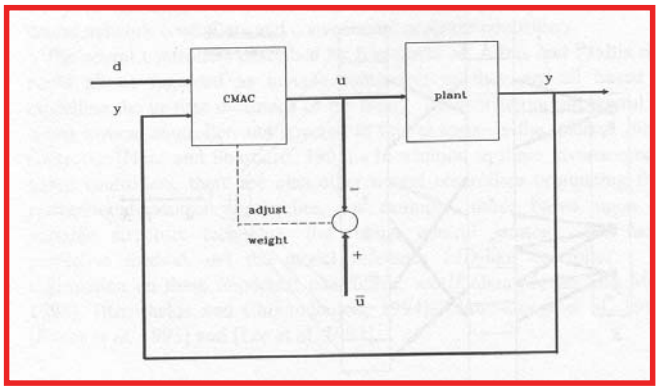
Figure 2.1 Input-output model

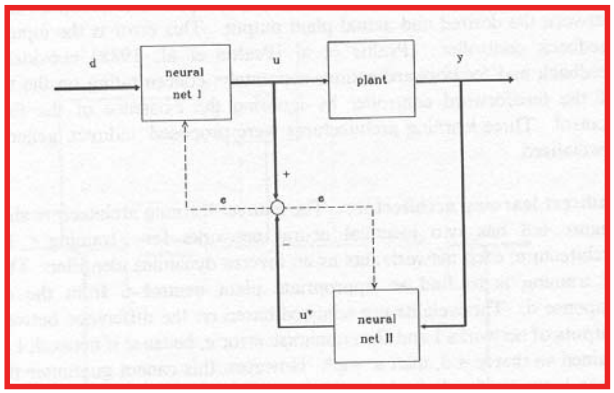




Funzione obiettivo: $y_p(k+1) = f(.)$
Funzione stimata: $y_{NET}(k+1) = F(.)$
Errore di previsione: $e(k+1)$

Controllo di Sistemi Non Lineari





d: *risposta desiderata/riferimento*
y: *uscita del sistema/ingresso di controllo*
u: *ingresso del sistema/uscita di controllo*
ū: *ingresso di controllo desiderato*
u^*: *uscita della rete neurale*
e: *errore di uscita e del controllo*

Lo scopo dell'addestramento della rete è quello di determinare un ingresso di controllo *u* utilizzando la risposta desiderata *d*. I pesi della rete sono stimati sulla base della differenza tra le uscite delle reti neurali I & II, e in modo da minimizzare **e**. Se la rete I è addestrata in modo tale che $y = d$, allora $u = u^*$. La rete agisce come stimatore della "dinamica inversa".

Reti Neurali per il Controllo

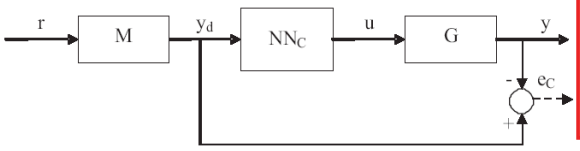


Figura 1: Controllo inverso diretto

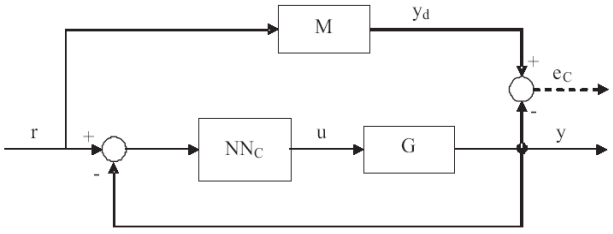


Figura 2: Controllo con modello di riferimento

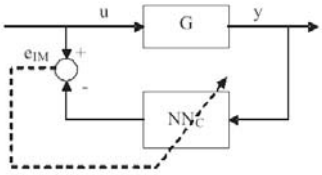


Figura 3: Addestramento della rete NN_C

Problemi delle Figure 1 e 3

- **Modelli instabili ad anello aperto**
- **Disturbi**

Controllo Adattativo con Modello Neurale di Riferimento (MRAC)

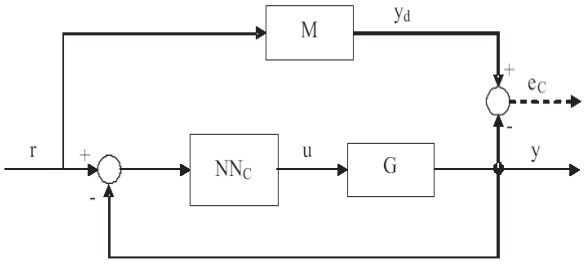


Figura 2: Controllo neurale con modello di riferimento

Il segnale e_C è utilizzato per l'apprendimento in linea dei pesi del controllore neurale NN_C. Sono 2 gli approcci impiegati per progettare un controllore MRAC per un processo con modello non noto: **Controllo Diretto e Indiretto**.

Controllo Diretto: Questo procedimento è in grado di determinare un controllore anche quando il modello dell'impiano non è disponibile. Dal momento che la conoscenza dell'impianto è necessaria per addestrare la rete neurale, rappresentata dal controllore NN_C, non è stato attualmente proposto nessun metodo.

Controllo Adattativo con Modello Neurale di Riferimento

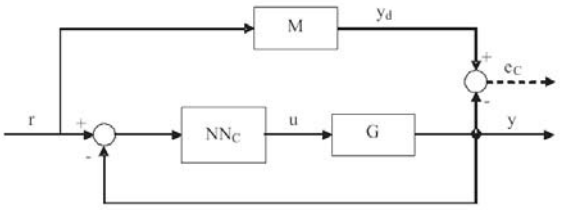


Figura 2: Controllo con modello di riferimento

Controllo indiretto

- Il segnale e_C viene usato per addestrare in linea i pesi del controllore neurale NN_C.

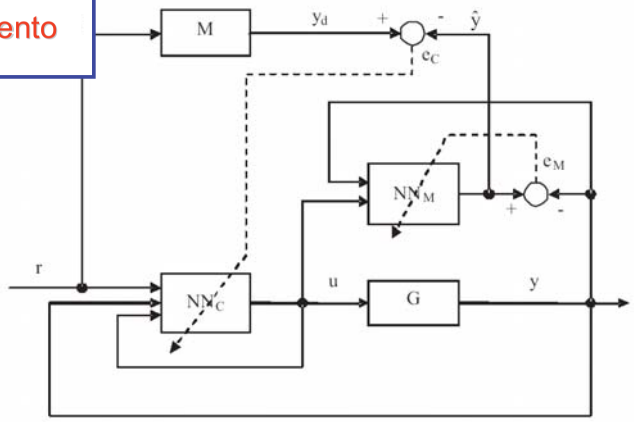


Figura 4: MRAC Indiretto

Controllo Indiretto: NN_M & NN_C

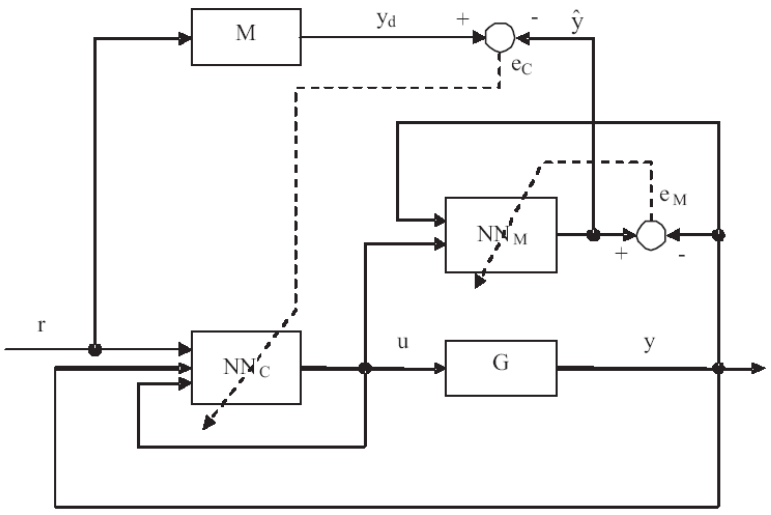


Figura 4: MRAC Indiretto

Questo metodo utilizza 2 reti neurali: la prima per la modellistica delle dinamiche del processo da controllare (NN_M), e la seconda viene addestrata per controllare il processo reale (G) in modo che il suo comportamento sia il più simile possibile a quello del modello di riferimento (M) con il controllore neurale (NN_C).

Controllo Indiretto (1)

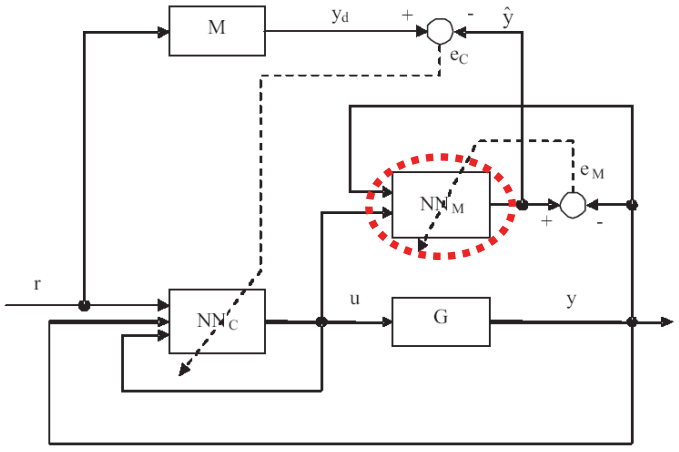


Figura 4: MRAC Indiretto

La rete neurale $\mathbf{NN_M}$ viene addestrata in modo da approssimare la relazione di ingresso-uscita del processo **G** e usando il segnale e_M. Questo viene solitamente effettuato fuori linea, usando un insieme di dati acquisiti dal processo ad anello aperto.

Controllo Indiretto (2)

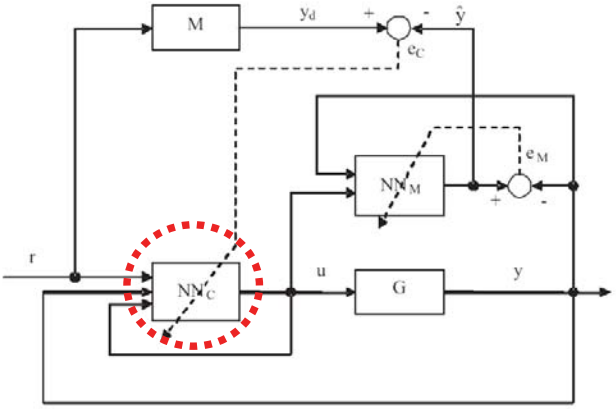


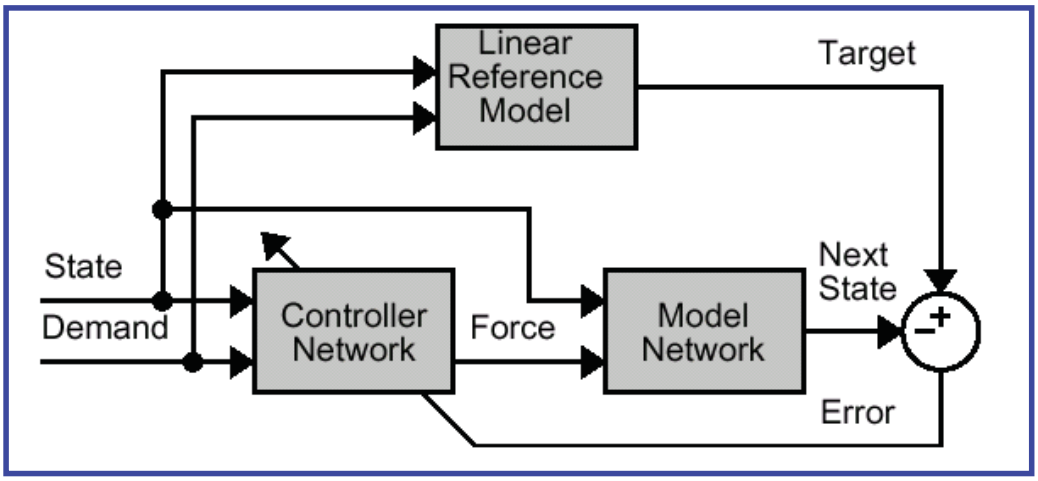
Figure 4: Indirect MRAC

NN_M è quindi fissata, e il suo comportamento è quindi noto e facile da calcolare.

Quando il modello NN_M è addestrato, viene utilizzato per allenare la rete NN_C che rappresenta il controllore. In genere viene impiegato il modello NN_M anziché l'uscita del processo reale perché l'impianto reale non è noto, e quindi non si può usare l'algoritmo di back-propagation. In queso modo, l'errore di controllo e_C è calcolato come differenza tra l'uscita del modello di riferimento $\hat{y}$ y_d e y, ovvero l'uscita prevista ad anello chiuso.

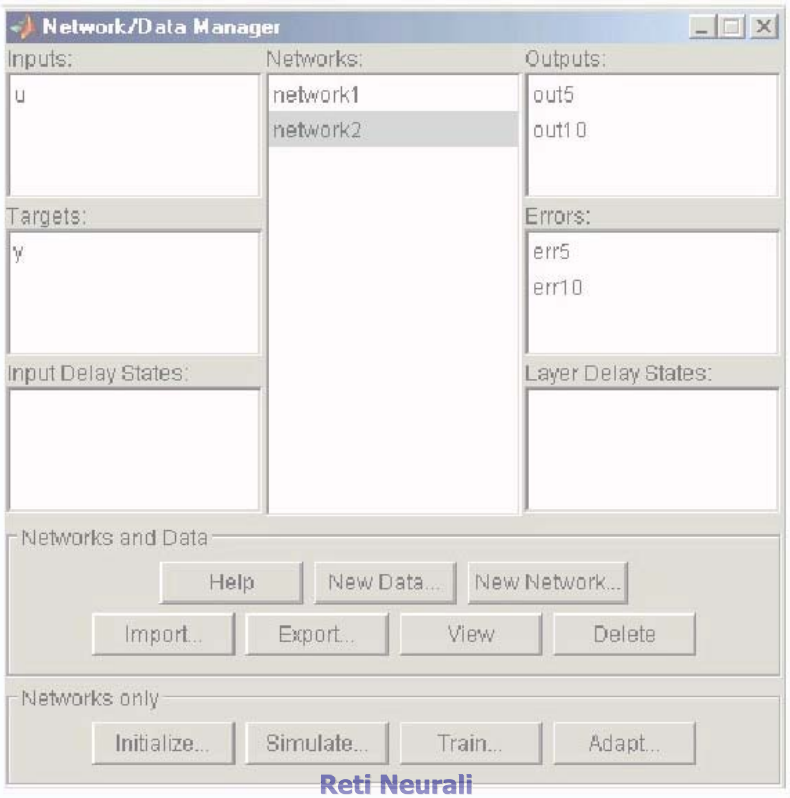
Controllo con Modello di Riferimento

Soluzioni in Matlab® e Simulink®

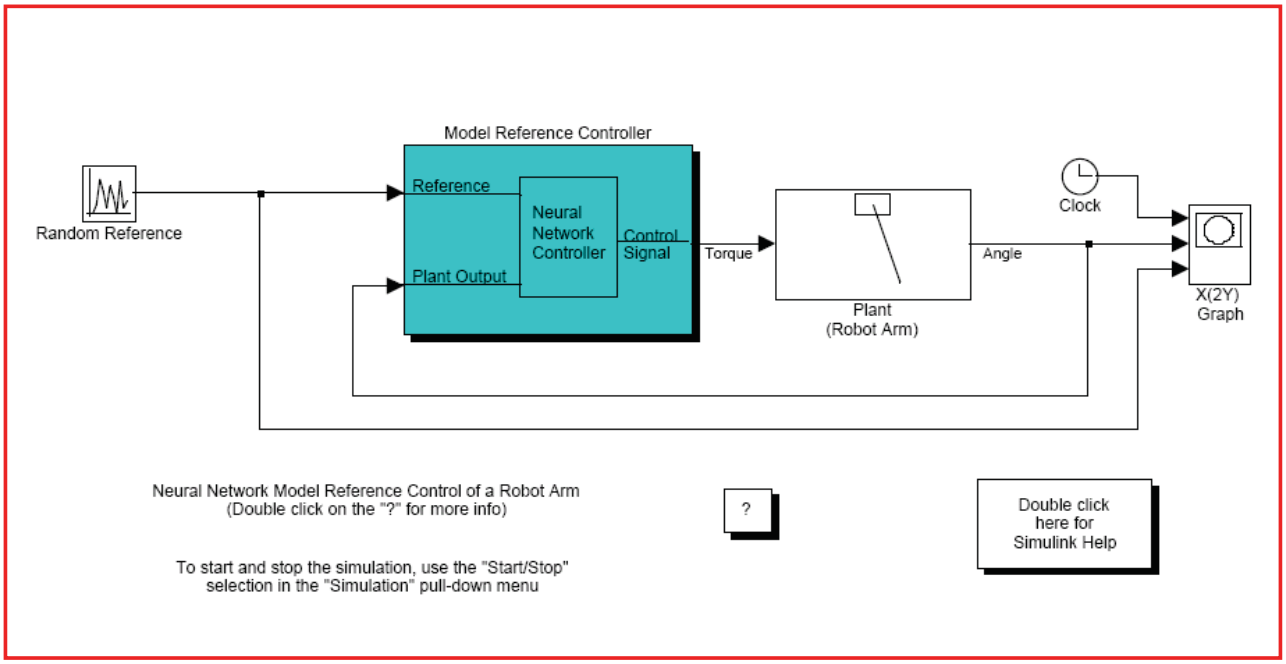


Controllore nurale, modello di riferimento e modello neurale

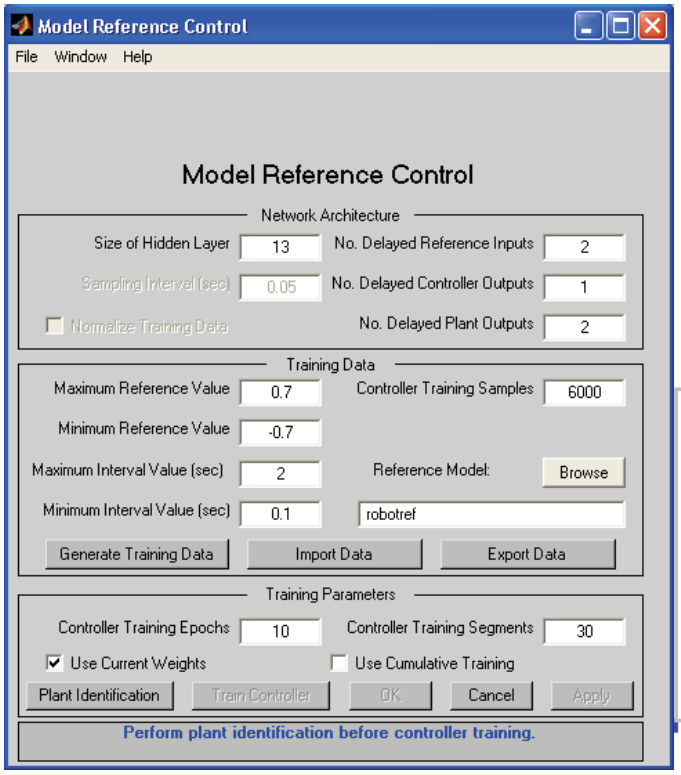
Matlab nntool GUI (Graphical User Interface)

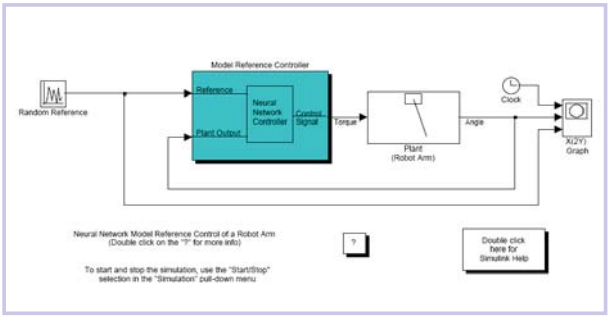


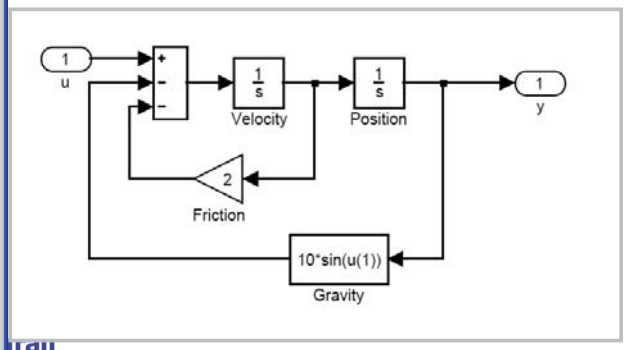
Esempio di Controllo di un Braccio di un Robot



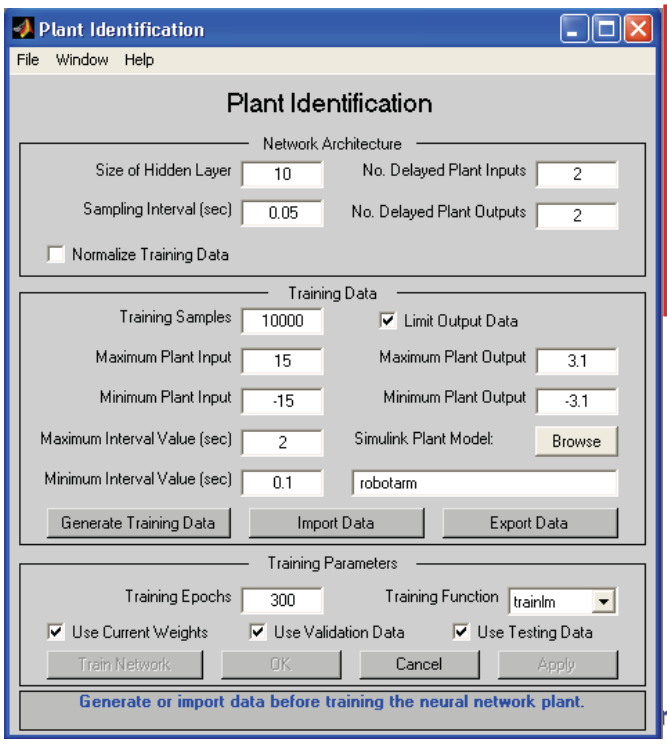
Controllo di un Braccio di un Robot: Esempio

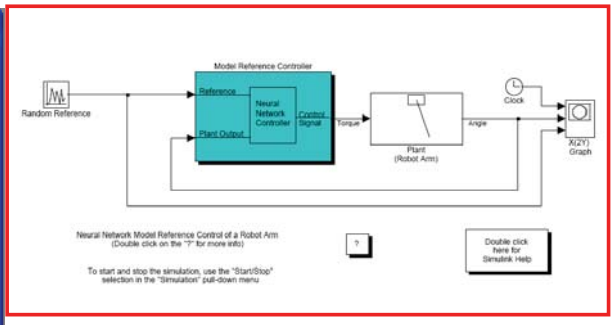






Controllo di un Braccio di un Robot (1)

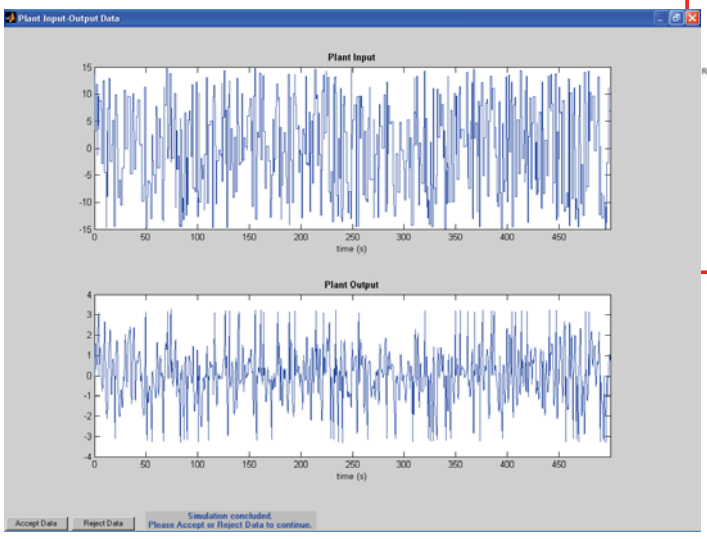


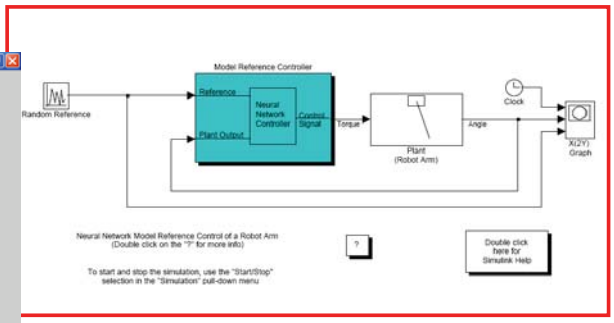


Identificazione del processo:

Generazione dei dati a partire dal modello di riferimento per l'addestramento della rete neurale

Controllo di un Braccio di un Robot (2)

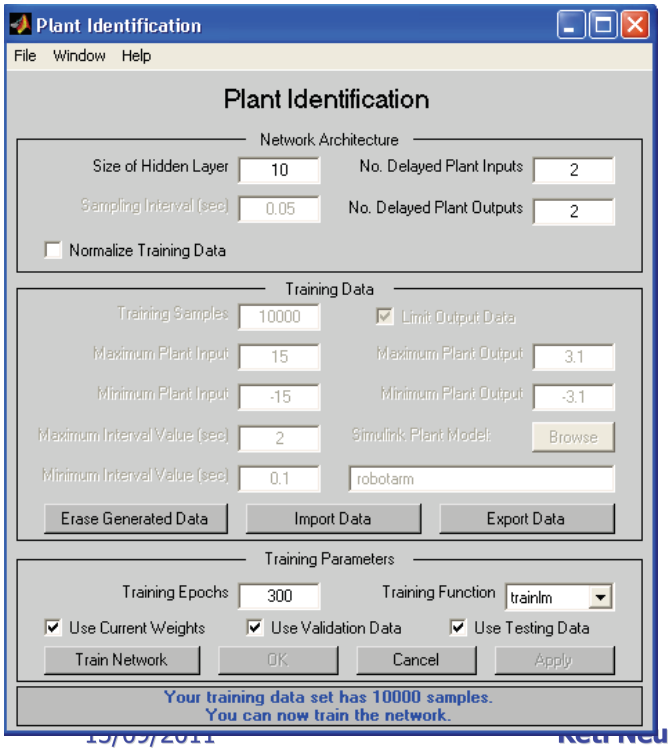


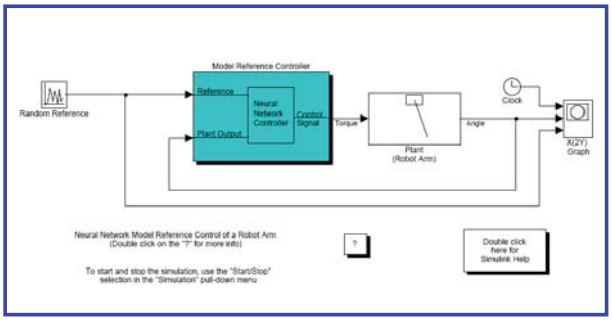


Dopo l'identificazione del processo:

Addestramento della rete neurale

Controllo di un Braccio di un Robot (3)

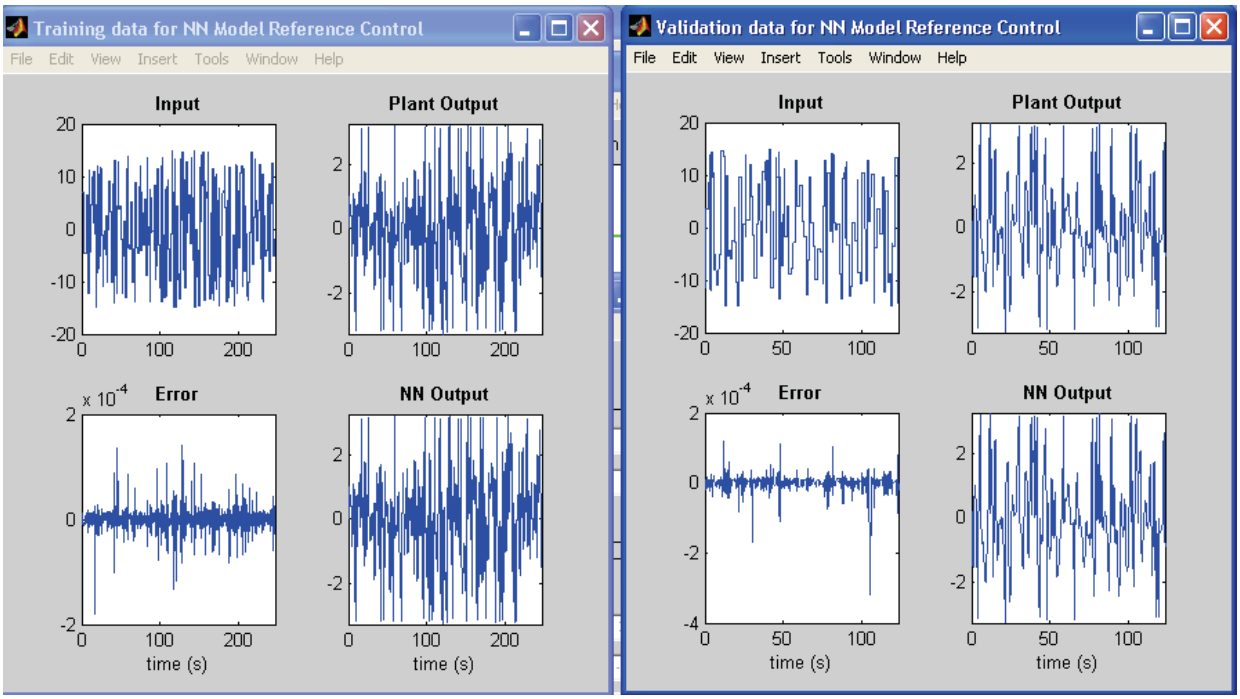




Dopo l'identificazione del processo:

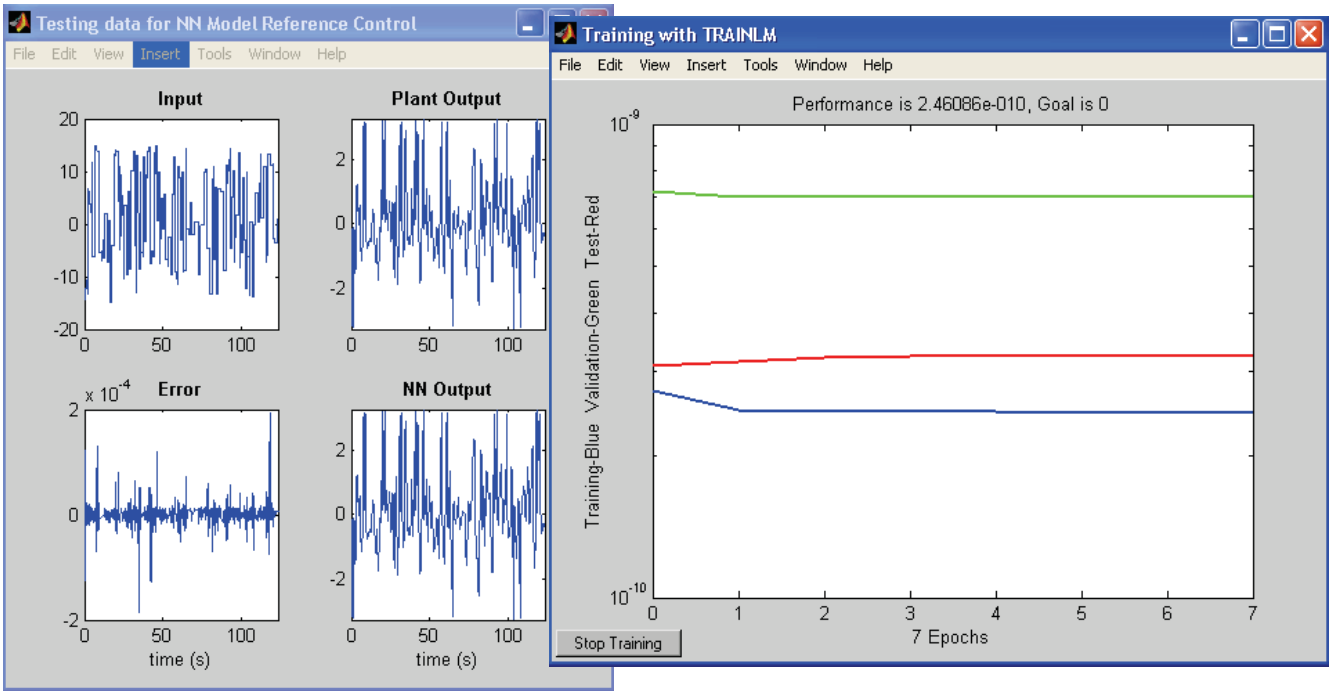
Addestramento della rete neurale

Controllo di un Braccio di un Robot (4)



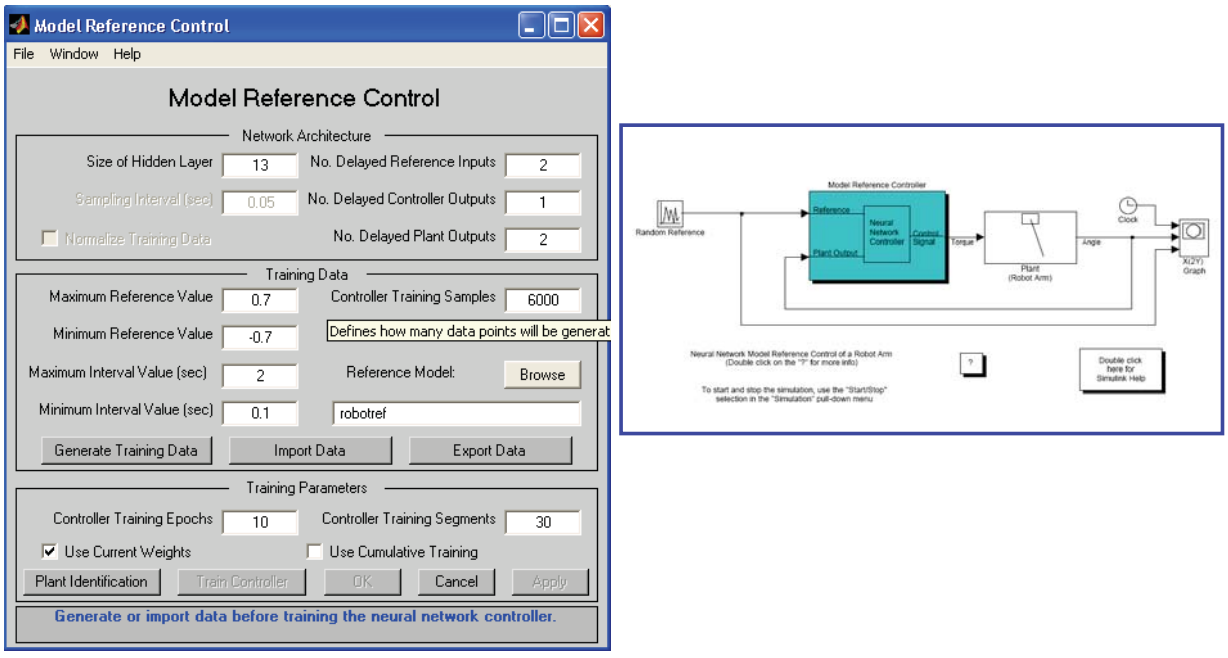
Dati per l'addestramento e la validazione

Controllo di un Braccio di un Robot (5)



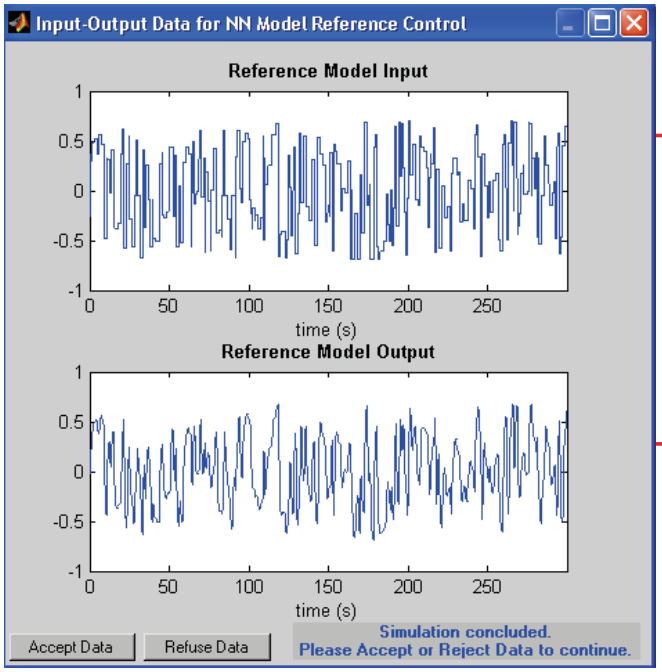
Dati di test e di risultati dell'addestramento

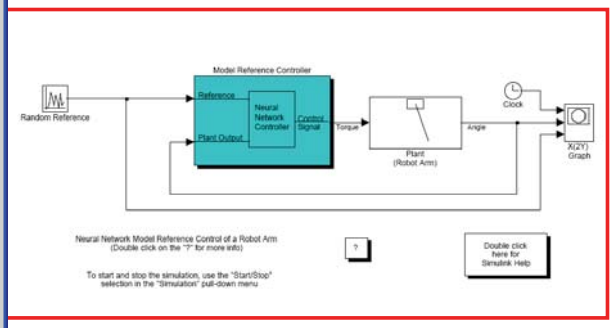
Controllo di un Braccio di un Robot (6)



Identificazione dell'impianto con una rete neurale: Generazione dei dati per l'identificazione del controllore neurale

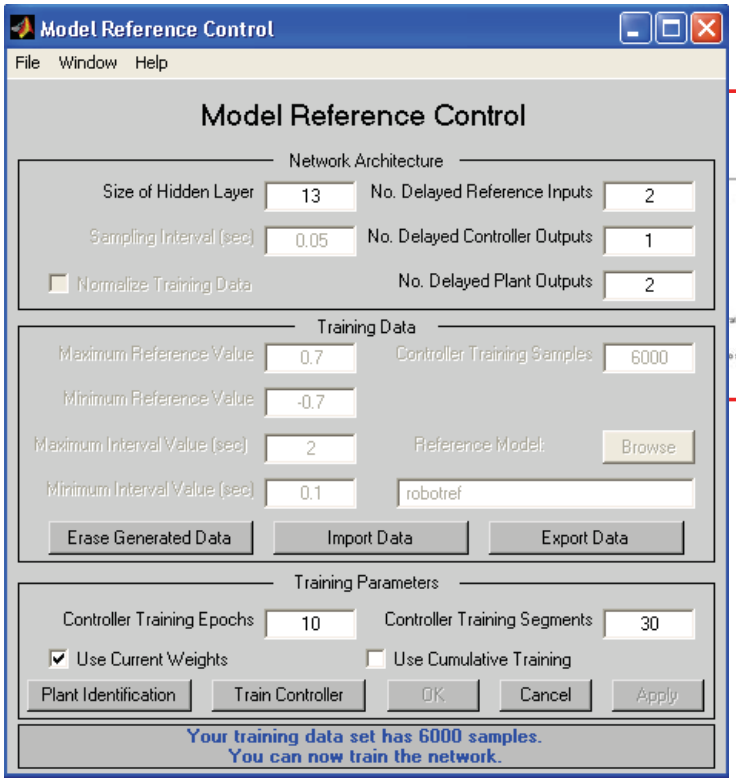
Controllo di un Braccio di un Robot (7)

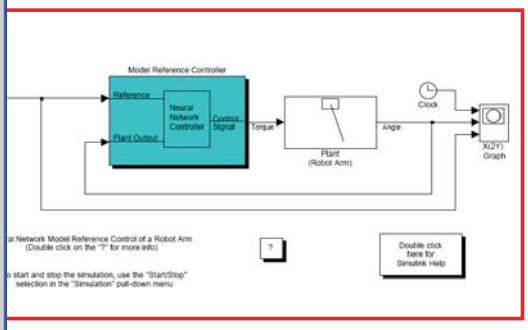




Riconoscimento dei dati che verranno impiegati per l'identificazione del controllore neurale

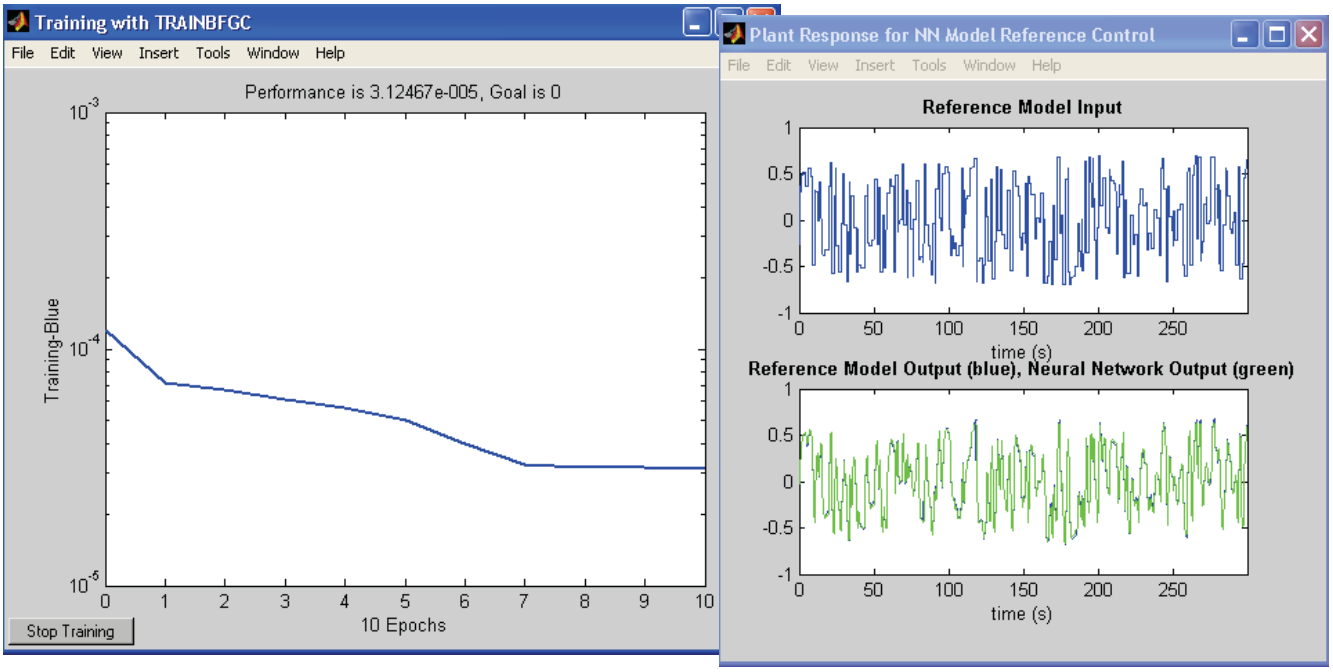
Controllo di un Braccio di un Robot (8)





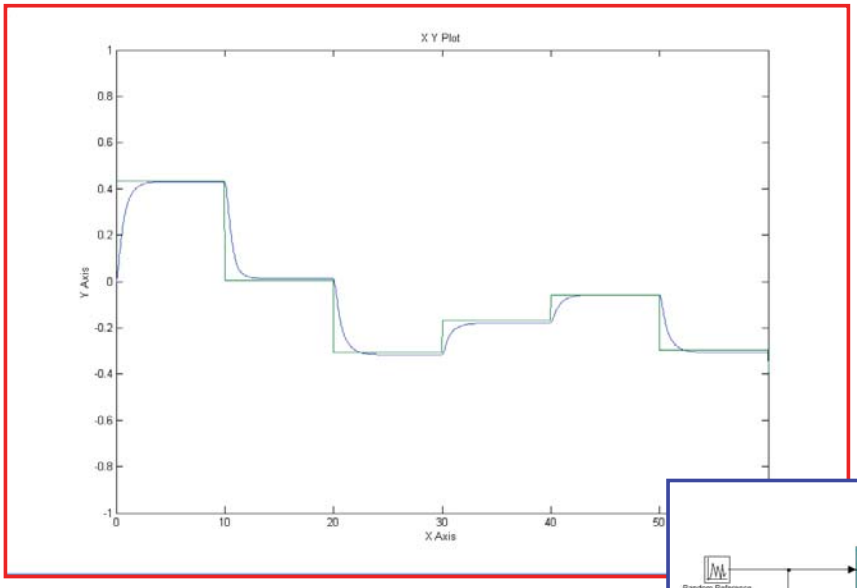
Addestramento del controllore neurale

Controllo di un Braccio di un Robot (9)

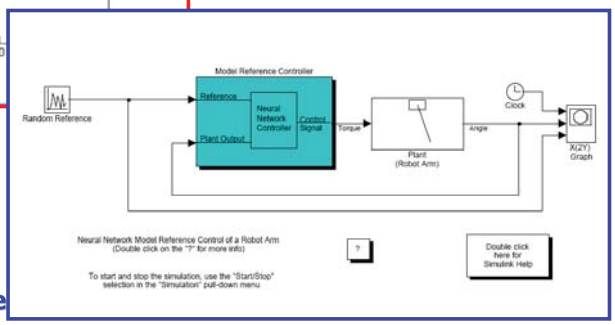


Addestramento del controllore neurale e risultati

Controllo di un Braccio di un Robot (10)



Segnale di riferimento e uscita controllata



Risultati della simulazione finale

Riferimenti Bibliografici

- Neural Networks for Identification, Prediction, and Control, by Duc Truong Pham and Xing Liu. Springer Verlag; (December 1995). ISBN: 3540199594.
- Nonlinear Identification and Control: A Neural Network Approach, by G. P. Liu. Springer Verlag; (October 2001). ISBN: 1852333421.
- Fuzzy Modeling for Control, by Robert Babuska. Springer; 1st edition (May 1, 1998) ISBN-10: 0792381548, ISBN-13: 978-0792381549.
- Multi-Objective Optimization using Evolutionary Algorithms, by Deb Kalyanmoy. John Wiley & Sons, Ltd, Chichester, England, 2001.