

APPRENDIMENTO SUPERVISIONATO

Reti multistrato

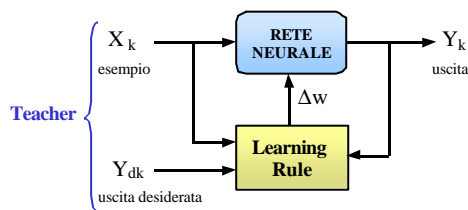
Apprendimento con supervisione

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

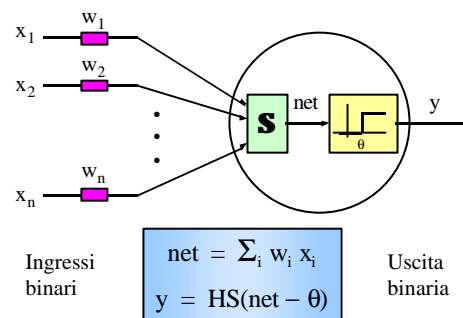
La rete opera in due fasi distinte:

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

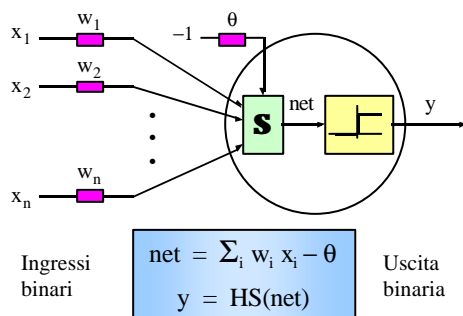
Fase di Addestramento



Il Perceptron (Rosenblatt '58)

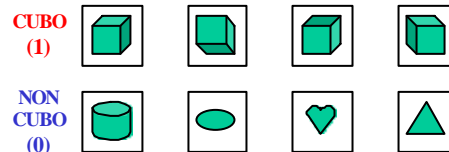


Il Perceptron (Rosenblatt '58)

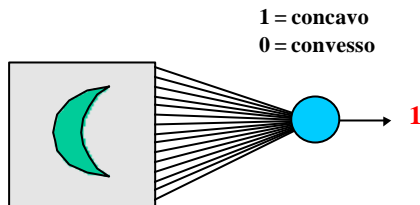


Classificazione

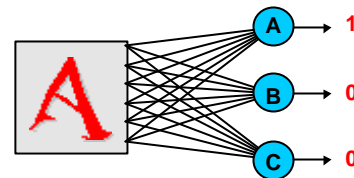
- Un perceptron può essere addestrato a riconoscere se un pattern d'ingresso X appartenga o no ad una classe C :



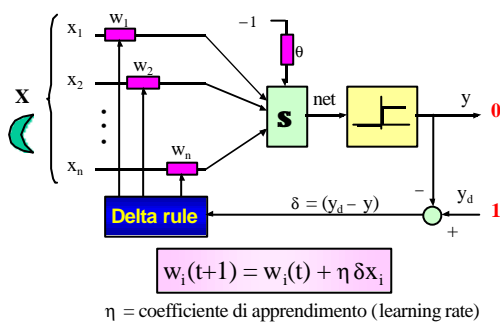
L'esperimento di Rosenblatt



Reti di Perceptron Riconoscitore di caratteri



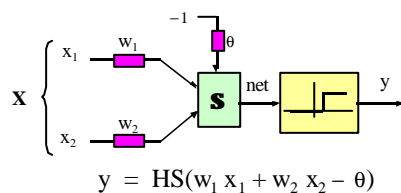
Addestramento



Algoritmo di apprendimento

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

Perceptron a due ingressi

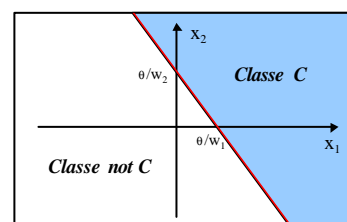


I pattern riconosciuti come appartenenti alla classe saranno quelli per cui:

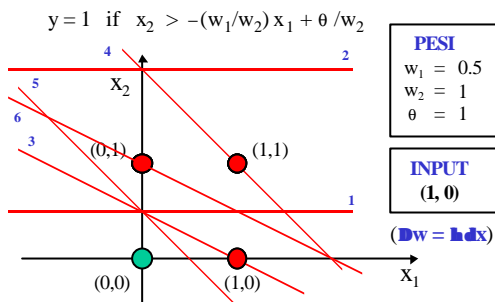
$$w_1 x_1 + w_2 x_2 - \theta > 0$$

Separazione lineare dello spazio d'ingresso

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



Apprendimento di un AND



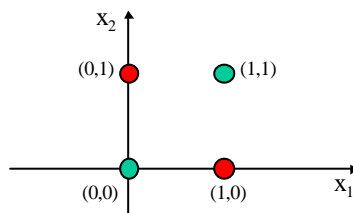
Limiti del Perceptron

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

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

Il problema dello XOR

Non è separabile linearmente!



Possibili soluzioni

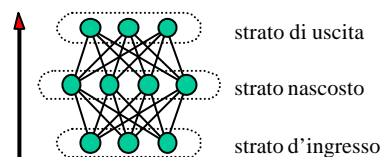
- Utilizzare neuroni con funzioni di uscita opportune.
- Combinare la risposta di più neuroni, secondo architetture multistrato.

Funzioni di uscita opportune

- $f(\text{net}) = \text{net}^2$, $w_1 = 1, w_2 = -1$
 $y = (x_1 - x_2)^2$
- $f(\text{net}) = |\text{net}|$, $w_1 = 1, w_2 = -1$
 $y = |x_1 - x_2|$
- $f(\text{net}) = 1 - e^{-|\text{net}|}$, $w_1 = 1, w_2 = -1$
 $y = 1 - e^{-|x_1 - x_2|}$

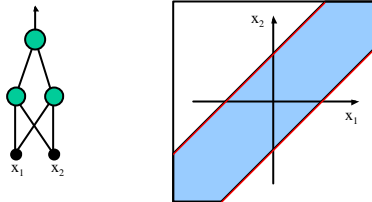
Reti multistrato

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



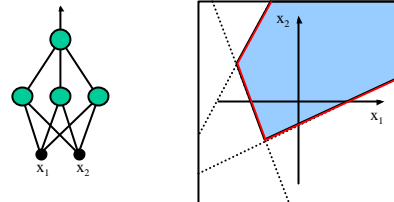
Reti a tre strati

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



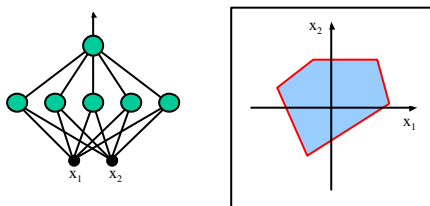
Reti a tre strati

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



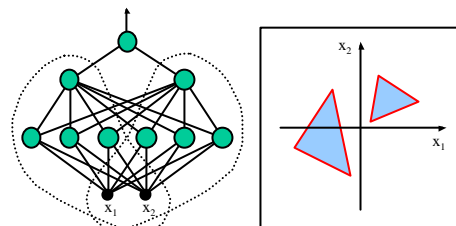
Reti a tre strati

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



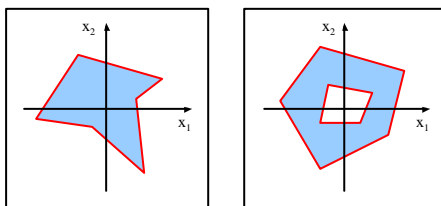
Reti a quattro strati

- Sono in gradi di separare regioni qualsiasi:



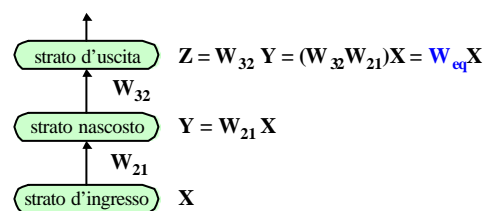
Reti a quattro strati

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



Importanza della non linearità

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



Implicazioni

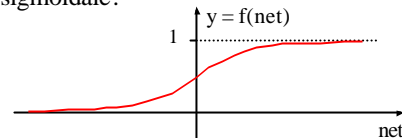
Per effettuare classificazioni complesse, i neuroni devono essere **non lineari** ed essere organizzati su **più strati**.

Problemi

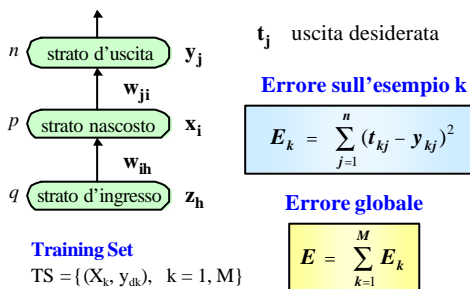
- Come si addestra una rete multistrato?
- Qual è l'uscita desiderata dei neuroni nascosti?

Back Propagation (Rumelhart-Hinton-Williams, '85)

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



Back Propagation: Definizioni



Errore sull'esempio k

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

Errore globale

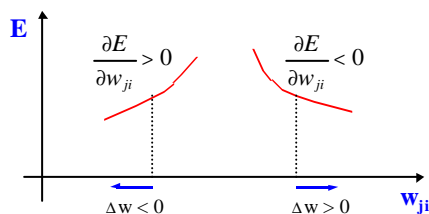
$$E = \sum_{k=1}^M E_k$$

Back Propagation: Obiettivi

- **Apprendimento**
insegnare alla rete un insieme di associazioni desiderate (X_k, t_k) : **Training Set (TS)**
- **Convergenza**
ridurre l'errore globale E al variare dei pesi, in modo che $E < \epsilon$
- **Generalizzazione**
far sì che la rete si comporti bene su esempi mai visti.

Convergenza

Per ridurre l'errore al variare dei pesi, si adotta la seguente strategia:



Aggiornamento dei pesi

Dunque i pesi sono variati in base alla seguente legge:

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

Regola del gradiente

η = coefficiente di apprendimento (learning rate)

Sviluppo con l'errore E_k

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

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

e posto $\delta_j = -\frac{\partial E_k}{\partial net_j}$ si ha: $\Delta w_{ji} = \eta \delta_j x_i$

Calcolo di δ_j (neuroni di uscita)

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

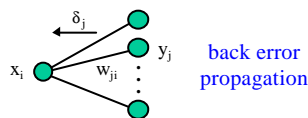
essendo $E_k = \sum_{j=1}^n (t_{kj} - y_{kj})^2$ si ha: $\frac{\partial E_k}{\partial y_j} = -(t_{kj} - y_{kj})$

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

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

Calcolo di δ_j (neuroni nascosti)

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



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

Aggiornamento dei pesi

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

Per lo strato di uscita:

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

Per lo strato nascosto

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

Back Propagation: Algoritmo

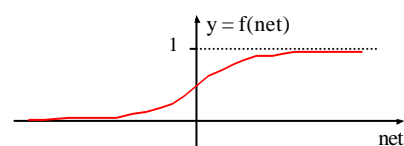
1. inizializza i pesi in modo casuale;
2. **do** {
3. inizializza l'errore globale $E = 0$;
4. **for each** $(X_k, t_k) \in TS$ {
5. calcola y_k e l'errore E_k ;
6. calcola i δ_j sullo strato di uscita;
7. calcola i δ_j sullo strato nascosto;
8. aggiorna i pesi della rete: $\Delta w = \eta \delta x$;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

Back Propagation: Osservazioni

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

net piccola $\rightarrow f'(net)$ grande $\rightarrow \Delta w$ grande

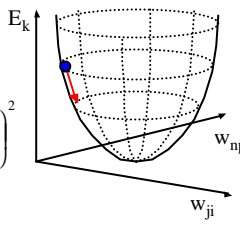
$$\Delta w = \eta \delta x \propto f'(net)$$



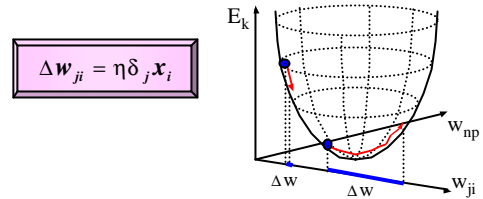
Back Propagation: Osservazioni

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

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

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


Back Propagation: Osservazioni



- troppo piccolo ► apprendimento lento
- troppo grande ► oscillazioni

Possibili soluzioni

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

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

■ è detto **momentum**

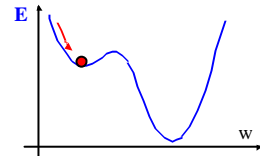
Back Propagation: Osservazioni

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

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

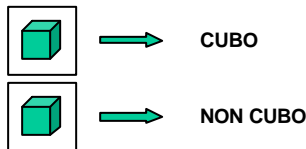
Rischio di fermarsi in un minimo locale

Occorre ricominciare da capo con nuovi pesi



Back Propagation: Osservazioni

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



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

Back Propagation: Osservazioni

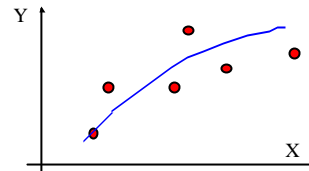
- Gli esempi del TS non sono altro che campioni di una funzione ingresso/uscita $F: \mathcal{R}^n \rightarrow \mathcal{R}^n$ che descrive il problema associativo.
- Durante l'apprendimento tale funzione viene approssimata mediante una combinazione non lineare di sigmoidi.
- Il processo di approssimazione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

Generalizzazione

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

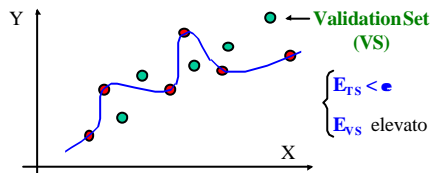
Generalizzazione

- Il numero di parametri da regolare dipende dal numero di neuroni nascosti della rete.
- Pochi neuroni nascosti potrebbero non essere sufficienti a ridurre l'errore globale:



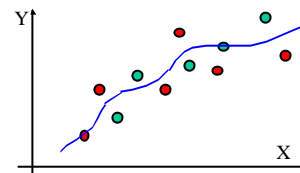
Generalizzazione

- Troppi neuroni nascosti potrebbero fossilizzare eccessivamente la rete sugli esempi specifici del TS.
- La rete risponderebbe bene sul TS, ma l'errore sarebbe elevato su altri esempi (**overtraining**).

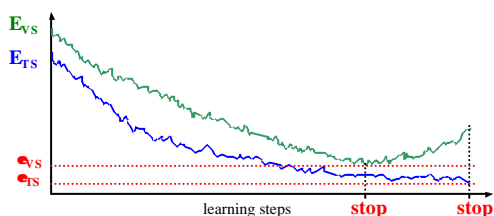


Generalizzazione

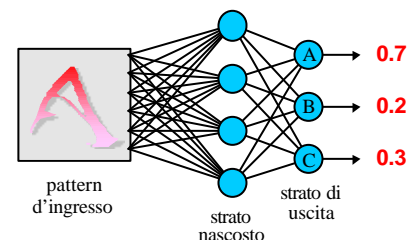
Per migliorare la capacità di generalizzazione si può addestrare la rete sul TS, monitorare l'errore sul VS (E_{VS}) e fermare l'apprendimento quando $E_{VS} < \epsilon_{VS}$:



Generalizzazione



Riconoscitore di caratteri

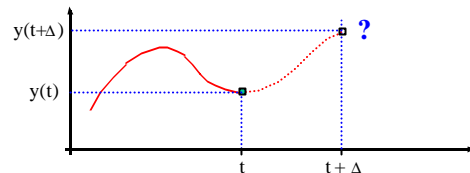


Altre applicazioni delle reti multistrato

Previsione di segnali

Problema

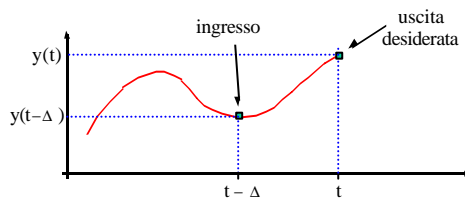
Imparare a prevedere il valore che un segnale (non casuale) avrà nel futuro sulla base del valore corrente:



Previsione di segnali

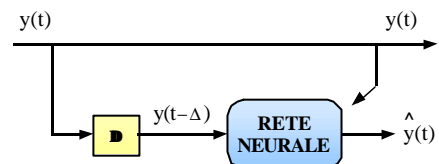
Metodo

Addestrare la rete sulla base dei valori passati.



Previsione di segnali

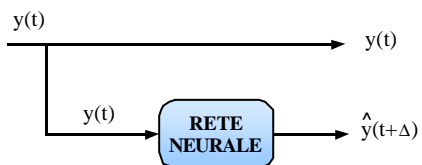
Fase di addestramento



La rete impara ad associare al valore $y(t-\Delta)$ il valore $y(t)$.

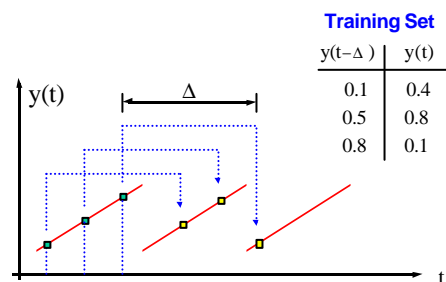
Previsione di segnali

Fase di esecuzione

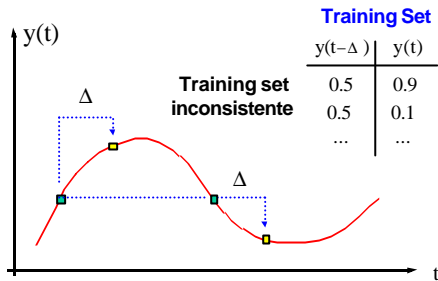


Ogni volta che si presenta il valore $y(t)$, la rete risponde con una stima di $y(t+\Delta)$.

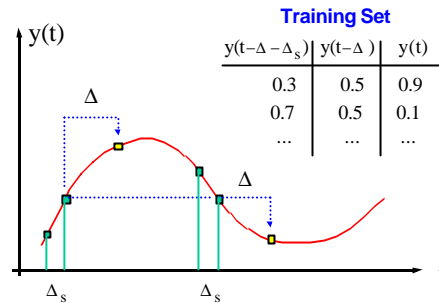
Esempio



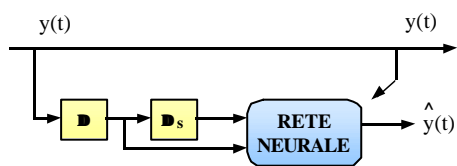
Problemi



Soluzione

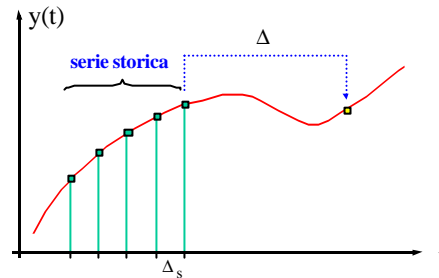


Addestramento

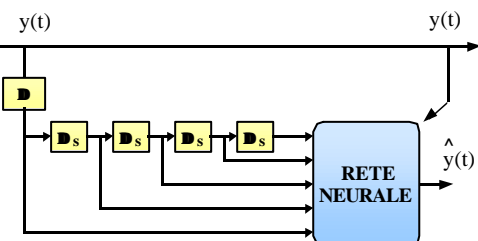


La rete viene addestrata con una coppia di valori passati: $y(t-\Delta-\Delta_s)$ e $y(t-\Delta)$.

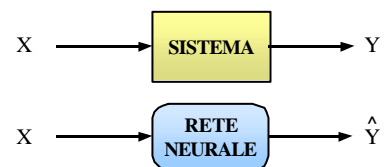
In generale



Addestramento



Identificazione

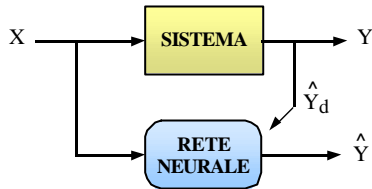


E' una fase essenziale delle tecniche di controllo:

- prima si identifica un modello del sistema;
- poi si costruisce una legge di controllo opportuna.

Identificazione (2)

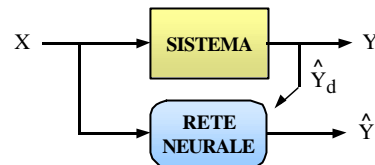
La rete può essere addestrata fornendo l'uscita del sistema come uscita desiderata.



Identificazione (3)

Tale schema di addestramento funziona solo se il sistema non possiede stati interni, ossia è del tipo:

$$Y = f(X)$$



Sistemi dinamici

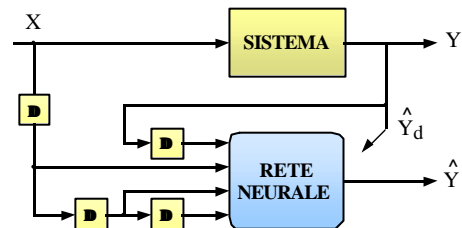
Sono sistemi in cui la risposta è funzione non solo dell'ingresso corrente, ma anche di uno stato interno $S(t)$.

Un sistema si dice di **ordine k** se la sua uscita dipende da al più **k** ingressi precedenti.

$$Y(t) = f[X(t-1), X(t-2), \dots, X(t-k)]$$

Addestramento

$$Y(t) = f[Y(t-1), X(t-1), X(t-2), X(t-3)]$$



Sistemi lineari

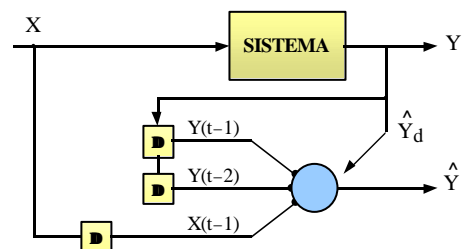
Sono sistemi in cui la risposta è una combinazione lineare della storia degli ingressi e delle uscite:

$$Y(t) = \sum_{k=1}^n a_k Y(t-k) + \sum_{k=1}^m b_k X(t-k)$$

Identificare il sistema significa stimare tutti i coefficienti a_k e b_k .

Esempio

$$Y(t) = a_1 Y(t-1) + a_2 Y(t-2) + b_1 X(t-1)$$



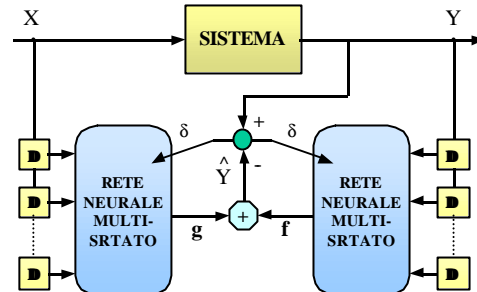
Sistemi non lineari

Sono sistemi in cui la risposta è una funzione non lineare degli ingressi e delle uscite:

$$Y(t) = f[Y(t-1), \dots, Y(t-n)] + g[X(t-1), \dots, X(t-m)]$$

Identificare il sistema significa stimare le funzioni **f** e **g**.

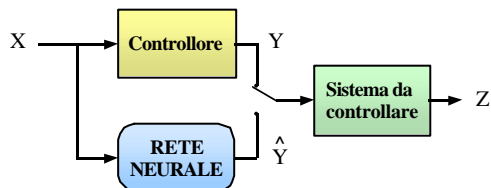
Addestramento



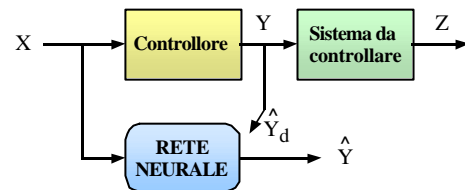
Copia di un controllore

Problema

Addestrare una rete neurale a comportarsi come un controllore esistente.



Copia di un controllore



Utile quando il controllore è molto costoso o poco pratico da usare.