Fabio Bicciato (c1b8)

RETI NEURALI

20 March 2009

Introduzione

I programmi tradizionali sono strumenti molto potenti in grado di svolgere una grande varietà di problemi esaminati e formalizzati in una adeguata procedura (algoritmo), ossia in una opportuna sequenza di passi che partendo da dati iniziali determinino la soluzione cercata in tempi ragionevoli.

La realizzazione di questi programmi prevede una fase di Analisi del problema, una fase di Programmazione attraverso la quale la procedura viene trasformata in programma, Test e quindi Manutenzione.

In questo modo la produzione di software specifico per la soluzione di un problema diventa onerosa ma soprattutto l'unica intelligenza presente nell'intero processo è quella degli analisti/programmatori.

Il dover determinare la soluzione in tempi ragionevoli determina inoltre l'impossibilità di tali sistemi di essere utilizzabili per la soluzione di problemi con algoritmi di grande complessità o per problemi di tipo combinatorio che richiedano la generazione ed il confronto di un numero troppo grande di soluzioni. Un esempio: il 'semplice' problema di assegnare N lavori L_i a N persone P_j che li eseguono con rendimento R_{ij}, volendo ottimizzare il rendimento globale si dovrebbero esaminare N! combinazioni, già con N=25 il numero di combinazioni possibili diventa astronomico.

Problemi basati sul ragionamento approssimato dove cioè i dati di input non sono sempre ben definiti, sono soprattutto qualitativi e di origine empirica, non sono sempre completi o contengono un certo grado di incertezza e per i quali non si conoscono procedure esatte per determinare la soluzione, sono inoltre difficilmente codificabili in programmi tradizionali.

Altra categoria di problemi non risolvibili con l'informatica tradizionale sono i problemi di tipo associativo dove cioè la soluzione dipende da precedenti associazioni dati/soluzione ottenute mediante sessioni di apprendimento. Esempi tipici sono il riconoscimento visivo/acustico di segnali e/o forme.

Come soluzione di tutte queste tipologie di problemi ci si accontenta spesso di una 'buona' soluzione, potrebbe non essere la migliore, ma sufficiente. L'individuazione nel minor tempo possibile di una buona soluzione con l'informatica tradizione potrebbe essere di enorme difficoltà da realizzare e per questo si sono sviluppati nel tempo diverse 'strategie' tra cui l'Intelligenza Artificiale, le Reti Neurali Artificiali, gli Algoritmi Genetici, ecc.

In questo articolo si introdurrà il neurone artificiale e quindi le Reti Neurali Artificiali Multistrato con algoritmo di apprendimento di tipo Error Back Propagation (EBP), il presente non vuole comunque essere una trattazione esaustiva dell'argomento ma solo una introduzione.

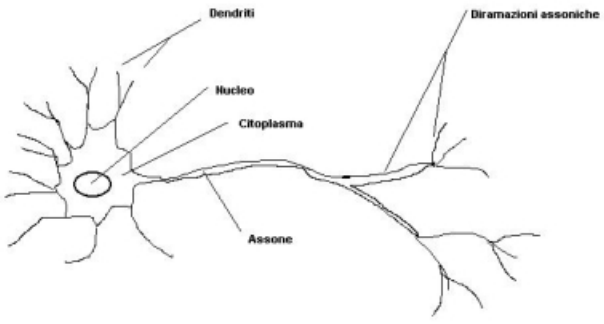
Le reti neurali artificiali possono essere implementate come software per personal computer, possono essere implementate in una FGPA o ancora in un sistema hardware/software (telefonini, palmari, ecc).

Gli utilizzi possibili sono svariati: riconoscimento vocale, riconoscimento immagini, movimentazione di un braccio robotico a 2 o più gradi di libertà, ottimizzazione utilizzo sistemi, guida automatica, ecc.

Requisito necessario è il corretto addestramento della rete stessa nello specifico problema di impiego.

Il Neurone Artificiale

Le reti neurali artificiali sono sistemi Hardware/Software che traggono ispirazione dalla neurofisiologia del cervello animale ed umano in particolare. Un neurone naturale può essere schematizzato come da figura:



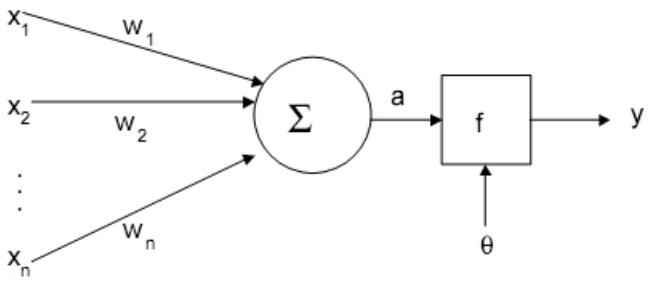
neurbio.JPG

Esso è composto da un corpo cellulare (Nucleo + Citoplasma) detto **Soma** messo in contatto (contatti **Sinattici** o *Sinapsi*) con una molteplicità di altri neuroni mediante le sue 'entrate': i **Dendriti**. Ogni contatto può essere inibitore o eccitatore, il neurone è reso *attivo* se la sua stimolazione globale supera una certa soglia che può variare nel tempo. Un neurone attivo emette un segnale che si propaga lungo l'**Assone** fino ai contatti sinattici di altri neuroni.

E' interessante notare che i neuroni sono estremamente più lenti dei componenti elettronici utilizzati in un calcolatore, un neurone commuta mediamente in un tempo dell'ordine del millisecondo. Si può quindi affermare che il funzionamento del cervello non dipenda dalla velocità di elaborazione di ogni neurone ma dalla numerosità dei neuroni (dell'ordine dei 100 miliardi) e dalla numerosità/complessità delle connessioni esistenti.

Caratteristica importate del cervello è che le connessioni tra neuroni non sono fissate ma si creano, si rinforzano o si indeboliscono con il tempo e nell'interazione con il mondo esterno o con altre persone.

Il **Neurone artificiale** (da ora neurone) è una semplificazione del neurone naturale:



Perceptron.jpg

Esso è costituito da n ingressi X (i dendriti) ai quali è associata un peso W (peso sinattico) ottenendo quindi degli ingressi pesati. Viene eseguita la somma degli ingressi pesati per ottenere il potenziale *a* del neurone. Il potenziale si applica alla funzione di trasferimento *f(a)* per ottenere il segnale *y* che viene trasmesso (assone) ai neuroni successivi o in uscita al sistema. La funzione di trasferimento f(a) ha inoltre in ingresso una soglia (*threshold*) θ che ne modifica il valore in ingresso.

Il semplice neurone appena visto può essere descritto dalla seguente:

(1) $y = f(a) = f(\sum_{i=1}^{n} x_i w_i - \theta)$

Interpretando la stessa soglia della funzione di trasferimento come un ingresso x_0 di valore -1 ed opportuno peso $w_0 = \theta$ si può scrivere la precedente nel modo seguente:

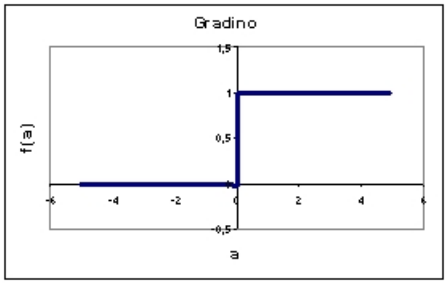
(2) $y = f(a) = f(\sum_{i=0}^{n} x_i w_i)$

Funzione di trasferimento

Le funzioni di trasferimento f(a) comunemente utilizzate sono le seguenti:

- A gradino

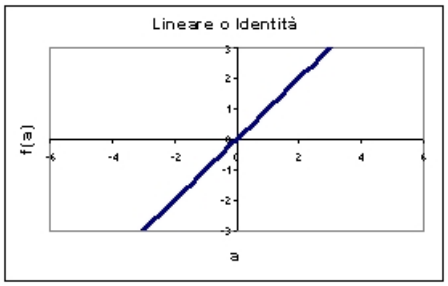
$$y = f(a) = \begin{cases} 1 & a > 0 \\ 0 (oppure -1) & a \leq 0 \end{cases}$$



Gradino.jpg

- Lineare o Identità

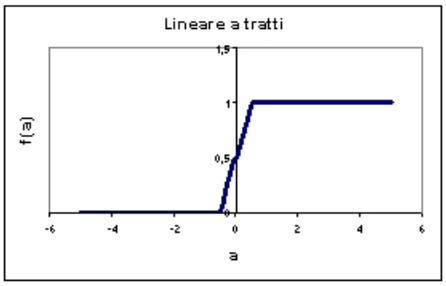
$y = a$



Lineare.jpg

- Lineare a tratti

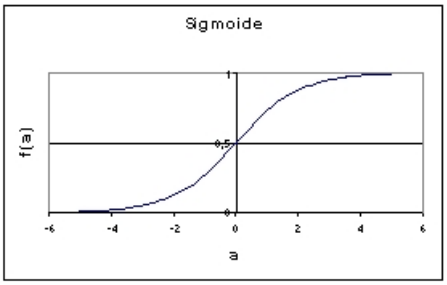
$$y = f(a) = \begin{cases} 0 & a \leq -0,5 \\ a + 0,5 & 0,5 < a < 0,5 \\ 1 & a \geq 0,5 \end{cases}$$



LineareTratti.jpg

- Logistica o Sigmoide

$$y = f(a) = \frac{1}{1 + e^{-a}}$$



Sigmoide.jpg

Un neurone con n ingressi pesati e funzione di trasferimento a gradino viene chiamato **Perceptron**.

Apprendimento

L'apprendimento di un neurone si svolge apportando modifiche ai pesi *w* in modo che il neurone stesso produca i risultati desiderati.

Diverse sono le regole utilizzabili per apportare tali modifiche, per semplicità nel seguito tratteremo solo la **Regola Delta Estesa** o **Regola di Widrow-Hoff**.

Uno dei metodi più utilizzati per l'addestramento è l'apprendimento supervisionato che prevede di presentare al neurone le informazioni di entrata e la corrispondente uscita. Al neurone saranno presentati più esempi (*training set*), sarà calcolato

l'errore quadratico globale, cioè la differenza tra l'uscita desiderata e l'uscita ottenuta dal neurone, dopo tutti gli esempi presentati, quindi apportate le opportune modifiche ai pesi w in ingresso.

Un ciclo di presentazione del training set e correzione dei pesi sinattici prende il nome di *epoca*. Al neurone saranno presentati gli stessi esempi per un numero sufficiente di epoche tale da rendere nullo o di valore minimo l'errore complessivo commesso.

Detta *k* il numero di esempi del training set, l'errore commesso all'esempio *k-esimo* è dato dalla:

$$E_k = \frac{1}{2}(y_k - t_k)^2$$

Dove t è il valore di uscita desiderato, y il valore di uscita del neurone. Si può quindi determinare la variazione da applicare ai pesi w_i, da:

$$\Delta w_i = \frac{\partial E}{\partial w_i}$$

Si ottiene:

(3) $$\Delta w_i = -\eta \sum_{k=1}^{n} [(y_k - t_k) f'(a_k) x_{ki}]$$

Dove t è il valore di uscita desiderato, y il valore di uscita del neurone, x_i il valore presente all'ingresso *i*, $f'(a)$ la derivata prima della funzione di trasferimento e η è il *learning rate* ossia un numero reale compreso tra 0 e 1 che determina la velocità di apprendimento del neurone. La (3) prende il nome di **Regola Delta Estesa**.

Per diminuire l'impiego di memoria impiegato da questo metodo si utilizza applicare la correzione ai pesi w del neurone non alla fine di ogni training set ma ad ogni esempio k applicato:

(4) $$\Delta w_i = -\eta (y - t) f'(a) x_i$$

Questo metodo è tuttavia meno preciso della variazione ottenuta cumulando l'errore di tutto il training set.

Per ottenere un migliore apprendimento è bene presentare esempi con risultati positivi e negativi nel dominio del problema.

Problemi lineari

Un problema di classificazione che separa in due classi i punti appartenenti ad un certo insieme si dice *lineare* se è possibile separare correttamente i punti mediante una retta (separazione in due dimensioni) o un iperpiano (in n dimensioni).

Un problema lineare in due dimensioni può essere facilmente risolto da un **Perceptron** con 2 ingressi, x_1 ed x_2, infatti esso è rappresentato (si veda la (1)) dalla:

$$y = x_1 w_1 + x_2 w_2 - \theta$$

Che per y=0 (una delle condizioni di uscita della funzione di trasferimento a gradino) si può scrivere:

$$x_2 = -\frac{w_1}{w_2} x_1 + \frac{\theta}{w_2}$$

La quale rappresenta una retta nel piano (x_1,x_2), retta di pendenza $-\frac{w_1}{w_2}$ ed intercetta $\frac{\theta}{w_2}$. Tale retta divide in due semipiani il piano (x_1,x_2), modificando i valori di w_1 e w_2 è possibile posizionare tale retta in modo che separi esattamente tutti i punti del problema appartenenti alle due diverse classificazioni.

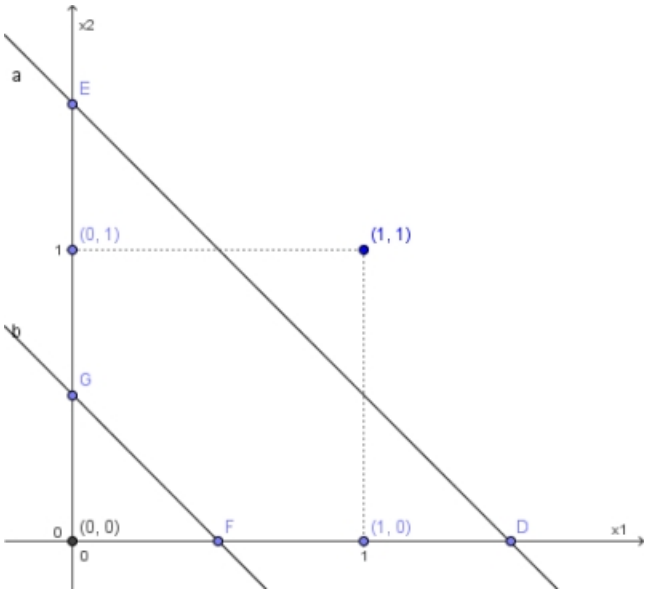
Così come un perceptron con 2 ingressi consente una separazione in due dimensioni un perceptron con n ingressi consente una separazione in n dimensioni.

Vediamo brevemente un esempio di separazione lineare in due dimensioni: funzione **OR** logico e funzione **AND** logico.

Detti x_1 ed x_2 i due ingressi possiamo descrivere i due problemi nel seguente modo:

x_1	x_2	**OR**	**AND**
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

Rappresentiamo le funzioni OR e AND in un piano (x_1,x_2) come di seguito:



OR_AND.jpg

Come si può notare sia la funzione OR che la funzione AND possono considerarsi problemi lineari in quanto esiste una retta (retta *a* per la funzione AND e retta *b* per la funzione OR) tale per cui i punti di uscita con valore '1' ed i punti di uscita con valore '0' siano ben separati.

Per risolvere la funzione AND è sufficiente quindi avere un perceptron che descriva la retta *a* ossia una retta di pendenza -1 e intercetta (punto E) maggiore di 1 e minore di 2, ad esempio:

$$w_1 = w_2 = 0,5 \text{ e } 1 < \frac{\theta}{w_2} < 2 \text{ da cui } \theta = 1,5 * 0,5 = 0,75$$

Allo stesso modo per risolvere la funzione OR si deve individuare una retta (la retta *b*) di pendenza -1 e intercetta (punto G) maggiore di 0 e minore di 1, ad esempio:

$$w_1 = w_2 = 0,5 \text{ e } 0 < \frac{\theta}{w_2} < 1 \text{ da cui } \theta = 0,5 * 0,5 = 0,25$$

I risultati appena ottenuti possono essere determinati in automatico mediante apprendimento con un algoritmo come il seguente (ponendo θ come peso di un ingresso di valore costante pari a -1):

1) Inizializzare i pesi w_1, w_2 e θ in modo casuale.

2) Presentare in ingresso le possibili combinazioni di x_1 e x_2

3) Per ogni combinazioni di ingressi calcolare l'uscita ottenuta dal perceptron e confrontarla con la corrispondente uscita desiderata.

4) Se necessario utilizzare la regola delta per correggere i pesi w_1, w_2 e θ

5) Ripetere dal punto 2) fino ad ottenere sempre risultati corretti.

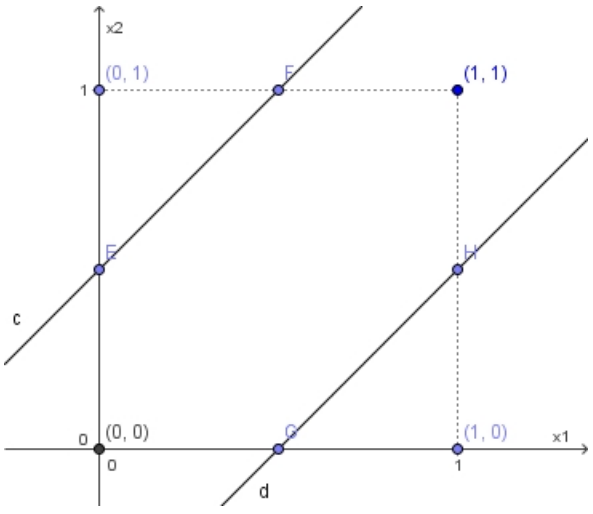
Reti multistrato

Abbiamo appena visto come un semplice perceptron sia in grado di risolvere problemi lineari in n dimensioni, ma per i problemi non lineari come possiamo fare?

Vediamo il problema non lineare **OR Esclusivo** (**XOR**).

Detti x_1 ed x_2 i due ingressi possiamo descrivere il problema nel seguente modo:

x_1	x_2	**XOR**
0	0	0
0	1	1
1	0	1
1	1	0



XOR_1.jpg

Riportando graficamente si nota che non è possibile separare con una sola retta i punti con uscita '1' (punti [0,1] e [1,0]) dai punti con uscita '0'.

E' però possibile utilizzare due rette per dividere il piano in 3 parti: due parti contenenti punti con soluzione '1' e una parte contenente i due punti con soluzione '0'.

Ogni retta è realizzabile con un perceptron. Avremo quindi:

- perceptron 1 (retta *c*) : retta di pendenza 1 con intercetta (punto E) maggiore di 0 e minore di 1,

$$w_1 = -w_2 = -0,5 \text{ e } 0 < \frac{\theta}{w_2} < 1 \text{ da cui } \theta = 0,5 * 0,5 = 0,25$$

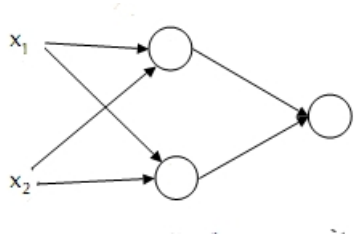
- perceptron 2 (retta *d*) : retta di pendenza 1 con intercetta (punto G) maggiore di -1 e minore di 0,

$$w_1 = -w_2 = 0,5 \text{ e } -1 < \frac{\theta}{w_2} < 0 \text{ da cui } \theta = -0,5 * -0,5 = 0,25$$

Considerando ora le uscite dei 2 perceptron come nuovi ingressi y_1 ed y_2 si ottengono le seguenti combinazioni (la combinazione 1,1 non è possibile perché i 2 neuroni non saranno mai contemporaneamente attivi):

y_1	y_2	**OUT**
0	0	0
0	1	1
1	0	1

Che diventa simile alla OR discussa in precedenza , abbiamo ora un problema lineare. In sostanza siamo riusciti a risolvere il problema XOR non lineare mediante l'impiego di 3 perceptron connessi come da figura:

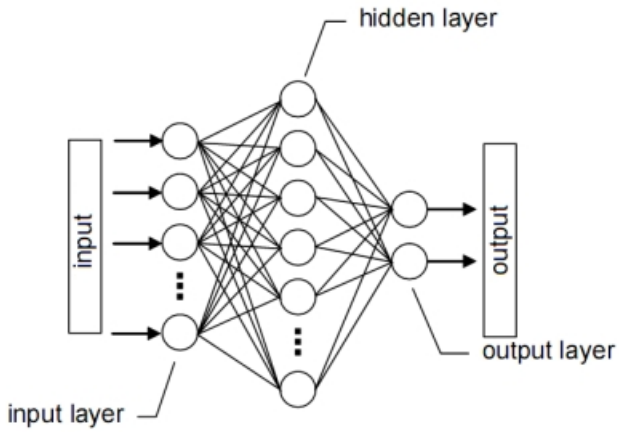


Rete_XOR.jpg

Quella appena realizzata è una rete multistrato ossia un insieme di neuroni disposti in ideali 'colonne' (**strati**) con ogni neurone di uno strato connesso in modo unidirezionale (riceve l'input dallo strato precedente e propaga l'output allo strato

successivo) con tutti i neuroni degli starti adiacenti ma non connesso con neuroni dello stesso strato o di strati non adiacenti.

Uno schema generale di una rete multistrato è il seguente:



Rete.jpg

La rete di figura presenta uno strato di input, uno di output ed uno strato 'nascosto' (ossia che non comunica direttamente con l'uscita). Gli strati hidden possono essere in numero variabile in funzione dello specifico problema da risolvere, ogni strato può avere un numero qualsiasi di neuroni, tuttavia è conveniente che gli strati hidden siano costituiti da un numero di neuroni maggiore sia dello strato di input che dello strato di output.

Una rete secondo cui il segnale viaggia da ingresso a uscita e non viceversa si chiama rete *feed-forward*.

Fondamentale nelle reti di neuroni è stabilire una *regola di attivazione* ossia l'ordine con cui i neuroni vengono elaborati. E' possibile scegliere tra *attivazione asincrona* (viene elaborato un neurone per volta) o *attivazione parallela* (tutti i neuroni elaborati contemporaneamente). Nella modalità di attivazione asincrona la scelta del neurone da elaborare può avvenire in modo casuale.

Esistono altre tipologie di reti, con connessioni tra neuroni dello stesso strato o di strati non adiacenti o ancora con connessioni bidirezionali, ma non tratteremo in questa sede queste reti neuronali.

Come per il semplice perceptron anche per le reti multistrato esiste, anzi si rende necessario, un algoritmo di apprendimento: **Error Back Propagation** (**EBP**).

Algoritmo EBP

Con la regola delta estesa vista in precedenza è possibile aggiornare solo lo strato di output della rete neurale, infatti solo per questo strato si è a conoscenza dei valori di output desiderati, valori confrontabili con i risultati della rete. Per tutti gli altri strati della rete i valori di output corretti non sono noti e devono in qualche modo essere determinati. Questo problema ha creato disinteresse nelle reti neuronali fino al 1986 quando si introdusse l'algoritmo **EBP**, con tale algoritmo è infatti possibile determinare i corretti valori di output per ogni neurone di uno qualsiasi degli strati.

EBP parte dal concetto che ogni neurone di uno strato nascosto *s* contribuisce, in maniera proporzionale al valore del proprio errore e al peso sinattico con i neuroni successivi, all'errore dello strato che lo segue.

Con questo presupposto è possibile, partendo dall'errore noto dell'ultimo strato di neuroni e procedendo a ritroso lungo gli strati della rete, determinare l'errore di uscita di ogni neurone.

Si può dimostrare che detto (si veda la (4)) $\delta_j = (y_j - t_j) * f'(a_j)$ l'errore del neurone *j* dello strato di output, è possibile determinare l'errore di un neurone *j* dello strato precedente con la seguente:

$$\delta_j = f'(a_j) \sum_{n=1}^{s} (\delta_n w_{jn})$$

Dove *s* è il numero di neuroni dello strato che trasmette all'indietro l'errore. Determinato quindi l'errore di ogni neurone di uno strato intermedio è possibile applicare la (4) ai neuroni dello strato stesso e procedere a ritroso con gli altri strati nascosti.

Per la determinazione dell'errore da propagare all'indietro è necessario derivare la funzione di trasferimento dei neuroni utilizzati, per questo motivo in una rete multistrato si preferisce utilizzare la funzione sigmoide che presenta la seguente derivata prima:

$$f'(a) = f(a)[1 - f(a)]$$

In conclusione potremmo descrivere l'algoritmo EBP nel seguente modo:

1) Si inizializzino tutti i pesi della rete con valori causali, non troppo alti.

2) Si presenti in ingresso un esempio con valori di ingresso x_k e valori di uscita t_k

3) Si calcolino le uscite o_k di tutti i neuroni dello strato di uscita della rete

4) Si determini l'errore di ogni neurone *j* dello strato di output e quindi si esegua la correzione dei pesi w_{ij} dei neuroni stessi:

$$\delta_j = (y_j - t_j) * f'(a_j)$$

5) Si determini l'errore di ogni neurone *j* dello strato precedente e si esegua la correzione dei pesi di ogni neurone:

$$\delta_j = f'(a_j) \sum_{n=1}^{s} (\delta_n w_{jn})$$

6) Si proceda a ritroso tra gli strati applicando il punto 5) fino all'ingresso della rete.

7) Ripetere dal punto 2) per tutti gli esempi del training set

8) Alla fine del training set si determini l'errore medio o globale ottenuto e si ripeta l'intero ciclo di addestramento fino ad ottenere un errore nullo o al di sotto di una soglia prefissata.

Conclusioni

Concludo questa introduzione alle reti neurali citandone i maggiori pregi e difetti.

Come difetto principale si può sicuramente segnalare la mancanza di una precisa formalizzazione della soluzione del problema e quindi l'impossibilità di riprodurre la soluzione stessa in algoritmi tradizionali, la conoscenza che la rete assume del problema viene infatti distribuita all'interno della rete stessa in modo non prevedibile.

La distribuzione della conoscenza all'interno dell'intera catena di neuroni è però anche un pregio in quanto garantisce alla rete una certa resistenza al rumore. Infatti la rete è in grado di produrre risultati corretti sia con valori in ingresso non precisi sia quando alcuni neuroni che la compongono smettono di funzionare o funzionano male.

In questo caso le prestazioni saranno ridotte ma la rete sarà comunque in grado di produrre soluzioni senza arrivare ad un blocco del sistema.

Altra importante proprietà delle reti neurali è la capacità di generalizzazione. Le reti sono in grado infatti di produrre soluzioni corrette e mai ottenute durante l'addestramento, al presentarsi di dati in ingresso mai incontrati durante la fase stessa di apprendimento.

Proprio per questa caratteristica le reti neuronali sono utilizzate in settori di previsione finanziario, meteorologico, ecc.

Fabio Bicciato

Estratto da "http://www.electroyou.it/mediawiki/index.php?title=UsersPages:C1b8:rete_neurale"