Introduzione alle Reti Neurali

Prof. Beatrice Lazzerini
Dipartimento di Ingegneria della Informazione
Via Diotisalvi, 2
56122 PISA

Reti Neurali Artificiali

Le reti neurali artificiali sono nate per riprodurre attività tipiche del cervello umano come la percezione di immagini, il riconoscimento di forme, la comprensione del linguaggio, il coordinamento senso-motorio, ecc.

A tale scopo si sono studiate le caratteristiche del cervello umano.

Nel sistema nervoso esistono miliardi di *neuroni* (cellule nervose). Un neurone è formato da un corpo cellulare e da molti prolungamenti ramificati, detti *dendriti*, attraverso i quali il neurone riceve segnali elettrici da altri neuroni. Ogni neurone ha anche un prolungamento filamentoso chiamato *assone*, la cui lunghezza può variare da circa 1 cm a qualche metro. All'estremità l'assone si ramifica formando terminali attraverso i quali i segnali elettrici vengono trasmessi ad altre cellule (ad esempio ai dendriti di altri neuroni). Tra un terminale di un assone e la cellula ricevente esiste uno spazio. I segnali superano questo spazio per mezzo di sostanze chimiche dette neurotrasmettitori. Il punto di connessione tra terminale e dendrite è detto *sinapsi*.

Un neurone si "attiva", cioè trasmette un impulso elettrico lungo il suo assone quando si verifica una differenza di potenziale elettrico tra l'interno e l'esterno della cellula. L'impulso elettrico provoca la liberazione di un neurotrasmettitore dai terminali dell'assone, che a loro volta possono, ad esempio, influenzare altri neuroni.

I neuroni biologici sono da 5 a 6 ordini di grandezza più lenti dei componenti elettronici convenzionali: un evento in un chip si verifica in alcuni nanosecondi (10^{-9} s) mentre un evento neurale in alcuni millisecondi (10^{-3} s).

Il cervello umano è un calcolatore complesso, non lineare e parallelo. Pur essendo costituito da elementi di elaborazione molto semplici (i neuroni), è in grado di eseguire computazioni complesse, come il riconoscimento, la percezione e il controllo del movimento, molte volte più velocemente del più veloce degli attuali calcolatori.

Il cervello è in grado di modificare le connessioni tra i neuroni in base all'esperienza acquisita, cioè è in grado di imparare.

Nel cervello non esiste un controllo centralizzato, nel senso che le varie zone del cervello funzionano insieme, influenzandosi reciprocamente e contribuendo alla realizzazione di uno specifico compito.

Infine, il cervello è *fault tolerant*, cioè se un neurone o una delle sue connessioni sono danneggiati, il cervello continua a funzionare, anche se con prestazioni leggermente degradate. In particolare, le prestazioni del processo cerebrale degradano gradualmente man mano che si distruggono sempre più neuroni (*graceful degradation*).

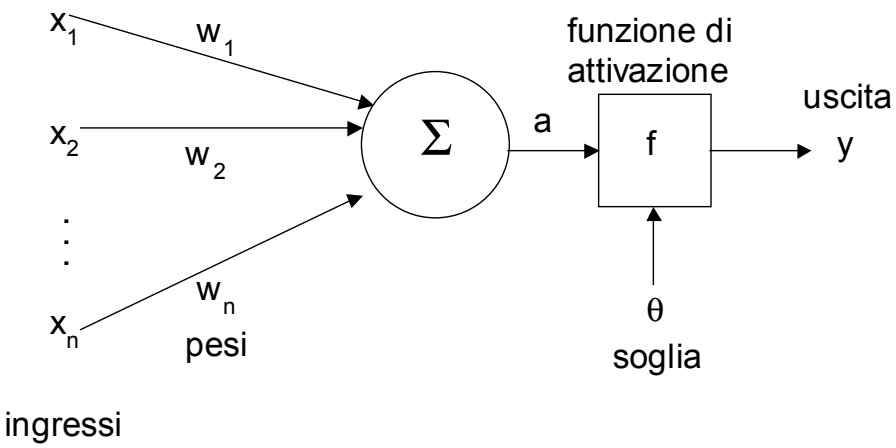
Quindi, per riprodurre artificialmente il cervello umano occorre realizzare una rete di elementi molto semplici che sia una struttura distribuita, massicciamente parallela, capace di apprendere e quindi di generalizzare (cioè produrre uscite in corrispondenza di ingressi non incontrati durante l'addestramento).

Tipicamente, il neurone artificiale ha molti ingressi ed una sola uscita. Ogni ingresso ha associato un peso, che determina la conducibilità del canale di ingresso. L'attivazione del neurone è una funzione della somma pesata degli ingressi.

Il metodo più usato per addestrare una rete neurale consiste nel presentare in ingresso alla rete un insieme di esempi (*training set*). La risposta fornita dalla rete per ogni esempio viene confrontata con la risposta desiderata, si valuta la differenza (errore) fra le due e, in base a tale differenza, si aggiustano i pesi. Questo processo viene ripetuto sull'intero training set finchè le uscite della rete producono un errore al di sotto di una soglia prestabilita.

Anche se di recente introduzione, le reti neurali trovano valida applicazione in settori quali predizione, classificazione, riconoscimento e controllo, portando spesso contributi significativi alla soluzione di problemi difficilmente trattabili con metodologie classiche.

MODELLO DI UN NEURONE



Abbiamo n canali di ingresso $x_1, \ldots, x_n$, a ciascuno dei quali è associato un peso. I pesi w_i sono numeri reali che riproducono le sinapsi. Se $w_i > 0$, il canale è detto *eccitatorio*, se $w_i < 0$, il canale è *inibitorio*. Il valore assoluto di un peso rappresenta la forza della connessione.

L'uscita, cioè il segnale con cui il neurone trasmette la sua attività all'esterno, è calcolata applicando la *funzione di attivazione* alla somma pesata degli ingressi. Indicando con $a = \sum_{i=1}^{n} w_i x_i$ la somma pesata degli ingressi, abbiamo:

$$y = f(a) = f\left(\sum_{i=1}^{n} w_i x_i\right).$$

Spesso, nella letteratura, la somma pesata degli ingressi è indicata con la parola *net*. Inoltre, la funzione di attivazione è detta anche *funzione di trasferimento*.

Nel modello di neurone rappresentato nella figura precedente è stata inclusa anche una *soglia* (*threshold*), che ha l'effetto di abbassare il valore in ingresso alla funzione di attivazione. Quindi, più correttamente, dobbiamo scrivere

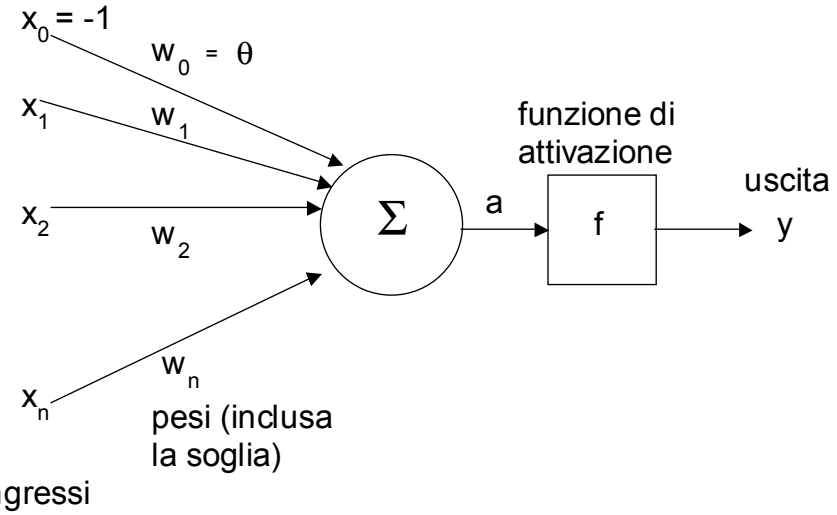
$$y = f(a) = f\left(\sum_{i=1}^{n} w_i x_i - \theta\right).$$

In questo caso, interpretando la soglia come il peso associato ad un ulteriore canale di ingresso x_0, di valore sempre costante e pari a -1, potremmo anche scrivere

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

con $w_0 = \theta$.

Il modello di un neurone diventa quindi:



Osserviamo che in alcuni casi, invece di considerare la soglia, si considera l'opposto della soglia, detto *bias*, che può quindi essere visto come il peso associato ad un ulteriore canale di ingresso di valore costante pari a 1. Avremo ancora

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

dove $x_0 = 1, w_0 = b$ (*b* è il bias).

Funzione di attivazione

La funzione di attivazione definisce l'uscita di un neurone in funzione del livello di attivazione $a = \sum_{i=0}^{n} w_i x_i$.

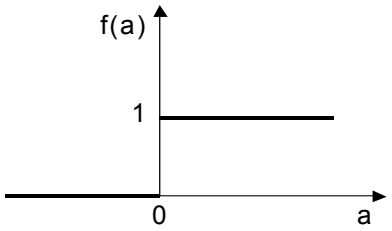
L'uscita può essere un numero reale, un numero reale appartenente ad un certo intervallo (ad esempio, [0,1]), oppure un numero appartenente ad un insieme discreto (tipicamente, {0,1} oppure {-1,+1}).

Vediamo alcuni esempi di funzione di attivazione.

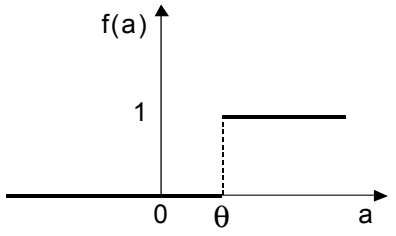
- **Funzione a soglia**

L'uscita di un neurone che usa una funzione di attivazione a soglia è

$$y = \begin{cases} 1 & \text{se } a \geq 0 \\ 0 & \text{se } a < 0 \end{cases}$$



Se si scorpora il contributo dell'ingresso x_0 dal livello di attivazione, cioè $a = \sum_{i=1}^{n} w_i x_i$, il diagramma diventa il seguente:

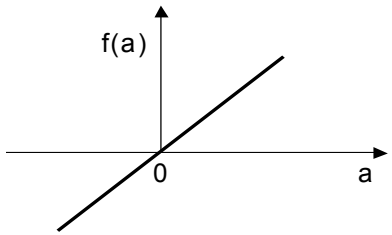


evidenziando così la notazione di soglia per il parametro θ:

$$y = \begin{cases} 1 & \text{se } \sum_{i=1}^{n} w_i x_i \geq \theta \\ 0 & \text{se } a < \theta \end{cases}$$

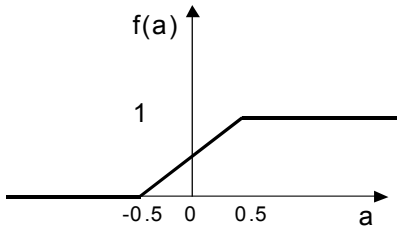
- **Funzione lineare**

f(a) = a



- **Funzione lineare a tratti**

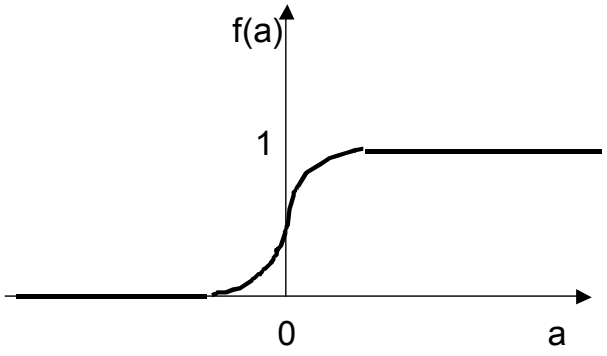
$$f(a)=\begin{cases} 0 & \text{se } a\le -0.5 \\ a+0.5 & \text{se } -0.5<a<0.5 \\ 1 & \text{se } a\ge 0.5 \end{cases}$$



- **Funzione sigmoide**

Assieme alla funzione di soglia, le funzioni sigmoidi sono tra le più usate. Un esempio di funzione sigmoide è la *funzione logistica*, definita come

$$f(a)=\frac{1}{1+\exp(-a)}$$



Osserviamo che, mentre una funzione a soglia assume solo i valori 0 e 1, una funzione sigmoide assume tutti i valori da 0 a 1. Notiamo, inoltre, che la funzione sigmoide è derivabile, mentre la funzione a soglia non lo è (questo ci servirà nel seguito).

Le funzioni di attivazione viste finora assumono valori tra 0 e +1 (esclusa la funzione lineare). A volte è opportuno che la funzione di attivazione assuma valori tra –1 e +1. In particolare, la funzione a soglia è ridefinita così :

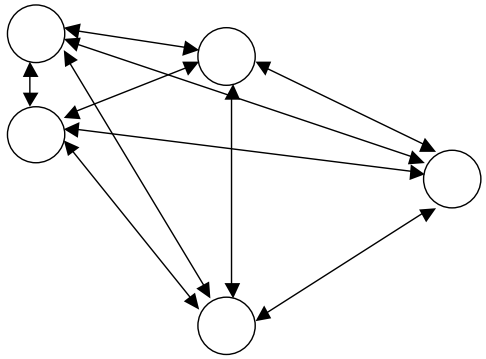
$$f(a)=\begin{cases} 1 & \text{se } a>0 \\ 0 & \text{se } a=0 \\ -1 & \text{se } a<0 \end{cases}$$

Tale funzione è nota come *funzione segno*.

ARCHITETTURA DI UNA RETE NEURALE

Si possono identificare più tipi di architettura di rete. Ne presentiamo due.

- **Reti completamente connesse (non stratificate)**



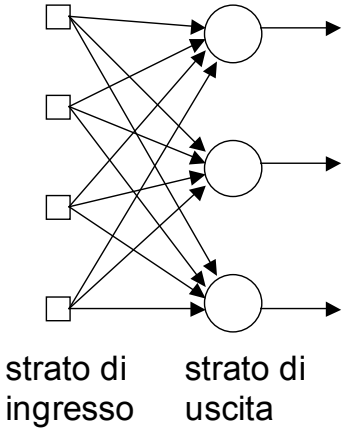
In una rete completamente connessa ogni neurone è connesso (in modo bidirezionale) con tutti gli altri.

Le connessioni tra i neuroni di una rete completamente connessa sono rappresentate mediante una matrice quadrata W, di dimensione pari al numero di neuroni, il cui generico elemento w_{ij} rappresenta il peso della connessione tra il neurone i ed il neurone j. Facciamo notare che alcuni autori usano, invece, la notazione w_{ji} per indicare il peso sulla connessione dal neurone i al neurone j.

- **Reti stratificate**

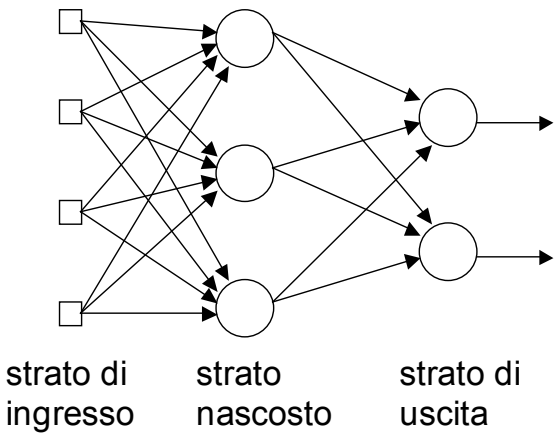
Nelle reti stratificate si individuano degli strati di neuroni tali che ogni neurone è connesso con tutti quelli dello strato successivo, ma non esistono connessioni tra i neuroni all'interno dello stesso strato, né tra neuroni di strati non adiacenti.

Il numero di strati ed il numero di neuroni per strato dipendono dallo specifico problema che si intende risolvere.



Dato che nello strato di ingresso non avviene alcuna computazione (i neuroni di ingresso devono semplicemente passare allo strato successivo i segnali ricevuti dall'ambiente esterno), la rete nella figura precedente viene di solito considerata come una rete con un solo strato. Inoltre, dato che i segnali viaggiano dallo strato di ingresso verso lo strato di uscita, si parla di rete *feedforward*.

Nella figura successiva viene mostrata una rete stratificata feedforward contenente uno *strato nascosto*, cioè uno strato i cui neuroni non comunicano direttamente con l'esterno. In generale, possono esserci uno o più strati nascosti. I neuroni nascosti permettono alla rete di costruire delle opportune rappresentazioni interne degli stimoli in ingresso in modo da facilitare il compito della rete.



Le connessioni tra i neuroni di una rete stratificata sono rappresentate mediante tante matrici quante sono le coppie di strati adiacenti. Ogni matrice contiene i pesi delle connessioni tra le coppie di neuroni di due strati adiacenti.

MODALITÀ DI ATTIVAZIONE DEI NEURONI

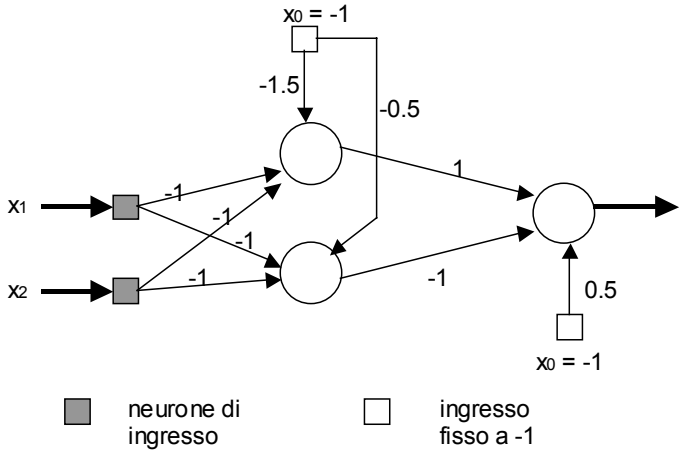
A seconda dei modelli di rete neurale, un solo neurone per volta può attivarsi ovvero tutti i neuroni possono attivarsi contemporaneamente. Nel primo caso si parla di *attivazione asincrona*, mentre nel secondo di *attivazione sincrona* o *parallela*. In particolare, nell'attivazione asincrona il neurone che può attivarsi è scelto in modo casuale.

Esempio

Costruiamo una rete neurale che calcola l'or esclusivo (XOR) che è così definito:

ingresso		
x1	x2	uscita
1	1	0
1	0	1
0	1	1
0	0	0

Consideriamo una rete con uno strato nascosto. Per calcolare l'uscita da ogni neurone dello strato nascosto e dello strato di uscita usiamo la funzione a soglia. (Ricordiamo che l'uscita di un neurone nello strato di ingresso coincide con il segnale in ingresso a tale neurone).



Diamo in ingresso alla rete la coppia [1 1]. Per il primo neurone nascosto con soglia -1.5 abbiamo:

$$a = (x_0 \times -1.5) + (x_1 \times -1) + (x_2 \times -1)$$
$$= (-1 \times -1.5) + (1 \times -1) + (1 \times -1) = -0.5$$

quindi l'uscita è 0. Per il secondo neurone nascosto con soglia -0.5 abbiamo:

$$a=(x_0\times -0.5)+(x_1\times -1)+(x_2\times -1)$$
$$=(-1\times -0.5)+(1\times -1)+(1\times -1)=-1.5$$

quindi l'uscita è 0. Per il neurone di uscita con soglia 0.5 abbiamo:

$$a=(x_0\times 0.5)+(x_1\times 1)+(x_2\times -1)$$
$$=(-1\times 0.5)+(0\times 1)+(0\times -1)=-0.5$$

quindi l'uscita è 0.

Facendo i calcoli con gli altri ingressi della tabella che definisce lo XOR, otteniamo le uscite indicate nella stessa tabella.

APPRENDIMENTO SUPERVISIONATO

Con riferimento all'esempio precedente, possiamo osservare che il corretto funzionamento della rete neurale dipende dall'architettura della rete (cioè dal numero di strati e dal numero di neuroni per strato), dalla funzione di attivazione dei neuroni e dai pesi. I primi due parametri sono fissati prima della fase di addestramento. Il compito dell'addestramento è quindi quello di aggiustare i pesi in modo che la rete produca le risposte desiderate.

Uno dei modi più usati per permettere ad una rete di imparare è l'*apprendimento supervisionato*, che prevede di presentare alla rete per ogni esempio di addestramento la corrispondente uscita desiderata.

Di solito i pesi vengono inizializzati con valori casuali all'inizio dell'addestramento. Poi si cominciano a presentare, uno alla volta, gli esempi costituenti l'insieme di addestramento (*training set*). Per ogni esempio presentato si calcola l'errore commesso dalla rete, cioè la differenza tra l'uscita desiderata e l'uscita effettiva della rete. L'errore è usato per aggiustare i pesi.

Il processo viene di solito ripetuto ripresentando alla rete, in ordine casuale, tutti gli esempi del training set finchè l'errore commesso su tutto il training set (oppure l'errore medio sul training set) risulta inferiore ad una soglia prestabilita.

Dopo l'addestramento la rete viene testata controllandone il comportamento su un insieme di dati, detto *test set*, costituito da esempi

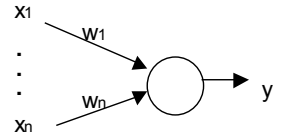
non utilizzati durante la fase di training. La fase di test ha quindi lo scopo di valutare la capacità di generalizzazione della rete neurale.

Diremo che la rete ha imparato, cioè è in grado di fornire risposte anche per ingressi che non le sono mai stati presentati durante la fase di addestramento.

Ovviamente le prestazioni di una rete neurale dipendono fortemente dall'insieme di esempi scelti per l'addestramento. Tali esempi devono quindi essere rappresentativi della realtà che la rete deve apprendere e in cui verrà utilizzata. L'addestramento è in effetti un processo ad hoc dipendente dallo specifico problema trattato.

Delta rule

Riferiamoci al neurone rappresentato di seguito:



La regola più usata per aggiustare i pesi di un neurone è la *delta rule* o *regola di Widrow-Hoff*. Sia $x = (x_1, \ldots, x_n)$ l'ingresso fornito al neurone. Se t ed y sono, rispettivamente, l'uscita desiderata e l'uscita neurale, l'errore δ è dato da

$$\delta = t - y.$$

La delta rule stabilisce che la variazione del generico peso Δw_i è:

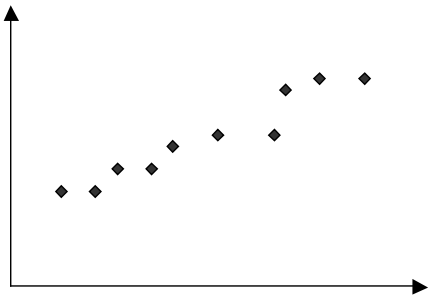
$$\Delta w_i = \eta \delta x_i$$

dove η è un numero reale compreso tra 0 e 1 detto *learning rate*. Il learning rate determina la velocità di apprendimento del neurone. La delta rule modifica in maniera proporzionale all'errore solo i pesi delle connessioni che hanno contribuito all'errore (cioè che hanno $x_i \neq 0$). Al contrario, se $x_i=0$, w_i non viene modificato poiché non si sa se ha contribuito all'errore. Il nuovo valore dei pesi è quindi:

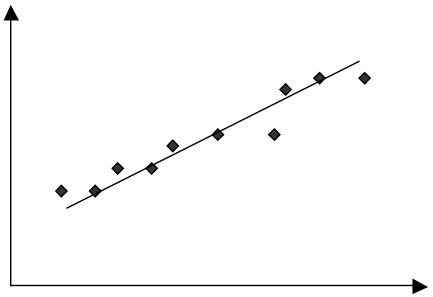
$$w_i = w_i + \Delta w_i$$

Esempio di addestramento

Molti problemi, per poter essere risolti, richiedono di adattare una retta o una curva ai dati che si hanno a disposizione. Consideriamo, ad esempio, un insieme di punti nel piano che seguono l'andamento di una linea retta ma non appartengono esattamente ad alcuna linea retta.



Possiamo interpolare i punti mediante una retta, calcolata, ad esempio usando il *metodo dei minimi quadrati*. Tale metodo ci permette di calcolare la retta che minimizza la somma degli errori quadratici per tutti i punti. Per ogni punto, l'errore è la distanza del punto dalla retta.



La retta disegnata può essere utile per ricavare (o prevedere) valori della variabile dipendente (rappresentata sull'asse delle ordinate) in corrispondenza di valori della variabile indipendente per cui non sono state fatte misurazioni.

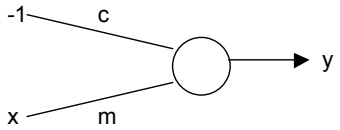
Considerando che l'equazione di una retta in forma esplicita è

$$y = mx + c$$

dove y ed x sono variabili, m la pendenza e c il punto in cui la retta incontra l'asse y, possiamo calcolare m e c con il metodo dei minimi quadrati risolvendo le equazioni ottenute uguagliando a 0 le derivate parziali (rispetto ad m e c, rispettivamente) della somma degli errori quadratici.

In alternativa al metodo dei minimi quadrati si può usare una rete neurale che approssima una retta. In ingresso alla rete vengono presentati i punti da approssimare con la linea retta e si lascia alla rete il compito di apprendere.

Possiamo usare una rete (rappresentata nella figura seguente) costituita da un neurone di ingresso ed un neurone di uscita con funzione di attivazione lineare. Dato che la rete deve stimare m e c, m e c costituiscono i pesi. Tali pesi saranno inizializzati in modo casuale. Gli esempi che costituiscono il training set sono le coppie (x, y) delle coordinate dei punti da approssimare con la linea retta; ovvero l'ascissa rappresenta l'ingresso e l'ordinata l'uscita desiderata. Il peso c è la soglia, quindi è associato ad un ingresso costantemente uguale a -1.



La rete viene addestrata usando la delta rule con un certo learning rate (ad esempio, $\eta = 0.1$). I punti del training set saranno presentati un certo numero di volte, finchè l'errore commesso dalla rete non scende al di sotto di una soglia prestabilita.

L'uscita della rete produrrà una retta del tutto simile a quella generata col metodo dei minimi quadrati.

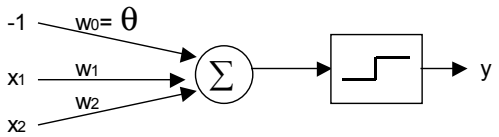
Diremo che la rete ha imparato perché dandole in ingresso l'ascissa di un punto non usato durante l'addestramento la rete produrrà in uscita l'ordinata corrispondente.

CLASSIFICAZIONE

In molte applicazioni si incontrano problemi di classificazione di un insieme di oggetti, cioè occorre associare ogni oggetto alla classe corretta.

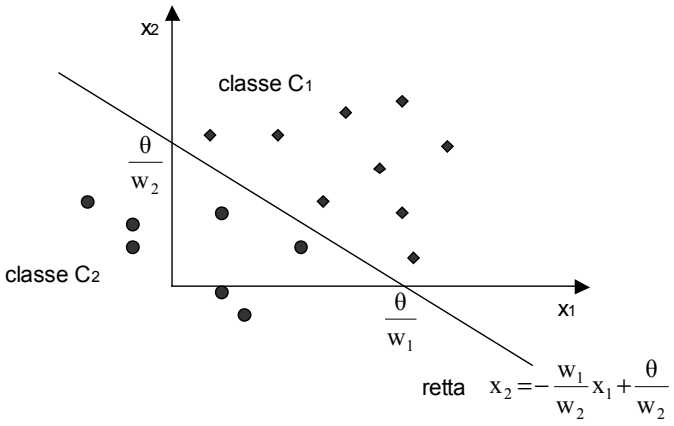
Supponiamo di voler classificare in due classi distinte oggetti rappresentati mediante punti nel piano. Se le due classi sono linearmente separabili, possiamo usare una rete neurale che approssima una retta di separazione tra le due classi. Un oggetto sarà quindi classificato rappresentandolo come punto nel piano ed assegnandolo a quella delle due classi individuata dal semipiano in cui cade il punto.

Tale classificazione può essere facilmente ottenuta addestrando una rete neurale con due ingressi ed un solo neurone di uscita con funzione di attivazione a soglia:



Tale rete è un esempio semplice di **perceptron**, costituito da più ingressi confluenti in un neurone di uscita con funzione di attivazione a soglia.

Consideriamo, ad esempio, la figura seguente:



Possiamo associare la classe C_1 all'insieme di stimoli per cui la rete risponde con y=1 e C_2 all'insieme di stimoli per cui la rete risponde con y=0, cioè:

$$x \in C_1 \quad \text{se} \quad y=1$$
$$x \in C_2 \quad \text{se} \quad y=0$$

Nel piano (x_1,x_2) degli ingressi della rete le classi C_1 e C_2 sono rappresentate da due semipiani separati dalla retta

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

Vediamo come è possibile addestrare il perceptron in modo che sia in grado di classificare correttamente i punti del piano. Dopo aver predisposto un opportuno training set, si fa uso della delta rule eseguendo i passi seguenti:

1. si inizializzano i pesi w_i con valori casuali;
2. si presenta alla rete un ingresso x_k insieme al valore t_k desiderato in uscita;
3. si calcola la risposta y_k della rete e si aggiornano i pesi mediante la delta rule;

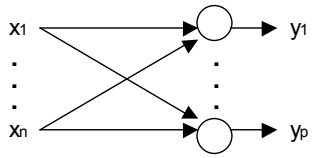
4. si ripete il ciclo dal passo 2, finchè la risposta della rete non risulti soddisfacente.

Con riferimento all'ultima figura, osserviamo che il processo di apprendimento, modificando i pesi w_1, w_2 e θ, non fa altro che modificare la posizione e la pendenza della retta di separazione tra le due classi. Il processo termina quando la retta separa correttamente le due classi.

Infine, osserviamo che la rete considerata nell'esempio è un perceptron con due ingressi perché gli oggetti da classificare sono rappresentati come punti in $\Re^2$. Tali punti sono separati da una retta. In generale, se gli oggetti da classificare sono vettori *n*-dimensionali, gli ingressi del perceptron sono *n* e le due classi sono separate da un iperpiano in $\Re^n$.

Aggiornamento dei pesi e convergenza della delta rule

Consideriamo la seguente rete con *n* ingressi e *p* uscite:



Siano t_j e o_j, rispettivamente, l'uscita desiderata e l'uscita effettiva del neurone *j*. L'errore E_k commesso dalla rete sull'esempio *k* può essere definito come:

$$E_k = \frac{1}{2}\sum_{j=1}^{p}(t_j - o_j)^2 \qquad (1)$$

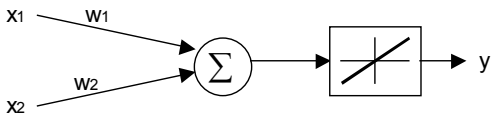
(il fattore ½ è stato introdotto per semplificare le notazioni)

per cui l'errore globale commesso dalla rete su tutto il training set costituito da *m* esempi è $E = \sum_{k=1}^{m} E_k$.

Per illustrare il razionale della delta rule, verrà preso in esame un caso semplice di comportamento lineare; anche se analiticamente tale situazione

può essere affrontata con metodi diretti, faremo riferimento alla procedura iterativa della delta rule.

Consideriamo allora un singolo neurone, rappresentato nella figura seguente, con due ingressi, soglia = 0 e funzione di attivazione lineare (cioè l'uscita coincide con l'ingresso: f(a)=a).



Tale neurone può modellare una qualunque linea retta passante per l'origine. Per un neurone lineare ed un singolo esempio, l'errore diventa:

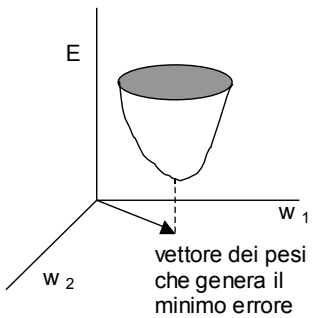
$$E = \frac{1}{2}(t-a)^2$$

Considerando che $a = x_1 w_1 + x_2 w_2$ e svolgendo i calcoli otteniamo:

$$E = \frac{1}{2}[t^2 - 2ta + a^2] = \\ = \frac{1}{2}[t^2 - 2t(x_1 w_1 + x_2 w_2) + x_1^2 w_1^2 + 2x_1 w_1 x_2 w_2 + x_2^2 w_2^2] \qquad (2)$$

Abbiamo ricavato che l'errore E, nel caso di neuroni lineari, è un paraboloide nello spazio dei pesi. In generale, per altri tipi di neuroni, sarà un'ipersuperficie nello spazio dei pesi.

Prima di iniziare l'addestramento, i pesi sono inizializzati a valori casuali quindi il punto che rappresenta lo stato iniziale della rete può trovarsi ovunque sulla superficie dell'errore (in generale non coinciderà con il punto di minimo di tale superficie). Durante l'addestramento i pesi dovranno essere modificati in modo da far muovere lo stato della rete lungo una direzione, che la delta rule individuerà essere quella di massima pendenza, della superficie dell'errore in modo da minimizzare l'errore globale.



Ricordiamo la definizione della delta rule con cui aggiustare i pesi:

$$\Delta w_{ij} = \eta \delta_j x_i \qquad \delta_j = (t_j - o_j)$$

dove t_j è l'uscita desiderata dal neurone j, o_j l'uscita effettiva, x_i il segnale proveniente dal neurone i, η il learning rate e Δw_{ij} la variazione del peso sulla connessione da i a j.

Vogliamo dimostrare che, aggiornando i pesi mediante la delta rule, l'apprendimento converge verso una configurazione dei pesi che minimizza l'errore quadratico globale.

Osserviamo che per dimostrare la convergenza della delta rule basterebbe dimostrare che tale regola è riconducibile alla forma

$$\Delta w_{ij} = -\frac{\partial E}{\partial w_{ij}}$$

la quale varierebbe i pesi in modo da favorire la diminuzione dell'errore. Infatti:

- se E cresce all'aumentare di w_{ij} (cioè $\frac{\partial E}{\partial w_{ij}} > 0$) allora w_{ij} viene <u>diminuito</u> per <u>contrastare</u> la crescita di E ($\Delta w_{ij} < 0$)
- se E diminuisce all'aumentare di w_{ij} (cioè $\frac{\partial E}{\partial w_{ij}} < 0$) allora w_{ij} viene <u>aumentato</u> per <u>favorire</u> la diminuzione di E ($\Delta w_{ij} > 0$).

Per semplicità, consideriamo un neurone lineare con uscita definita da

$$o_j = \sum_i x_i w_{ij}$$

Esprimiamo la derivata dell'errore rispetto ad un peso come prodotto di due quantità, la prima delle quali esprime il cambiamento dell'errore in funzione dell'uscita di un neurone, la seconda riguarda il cambiamento dell'uscita rispetto ad un peso:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial w_{ij}}$$

Dalla (1) e dalla definizione di δ_j otteniamo

$$\frac{\partial E}{\partial o_j} = -\delta_j$$

Inoltre $\frac{\partial o_j}{\partial w_{ij}} = x_i$. Di conseguenza

$$-\frac{\partial E}{\partial w_{ij}} = \delta_j x_i$$

A questo punto, inserendo il learning rate, otteniamo proprio la delta rule.

Con riferimento alla figura che mostra la superficie dell'errore, osserviamo che il processo di apprendimento può essere interpretato come una discesa su tale superficie lungo la linea di massima pendenza, individuata appunto dal gradiente $-\nabla E = \left(-\frac{\partial E}{\partial w_{ij}} \right)$.

Il learning rate η rappresenta quindi la rapidità di discesa di E sulla superficie. È importante scegliere il valore giusto per η: un valore troppo piccolo può comportare un apprendimento troppo lento, mentre un valore troppo elevato può provocare oscillazioni dell'errore intorno al minimo. La soluzione tipicamente adottata è quella di stabilire un valore alto di η (prossimo a 1) all'inizio dell'addestramento e diminuire tale valore mano a mano che si procede con l'apprendimento.

Osservazione

La delta rule, che abbiamo ricavato riferendoci per semplicità ai neuroni lineari, è in realtà valida per qualsiasi tipo di neurone. Si parla, in effetti, di delta rule nel caso di neuroni lineari e di *delta rule generalizzata* per altri tipi di neuroni, tenendo però conto della eventuale forma della funzione dell'errore da minimizzare.

Problemi lineari e non lineari

Un problema di classificazione in cui si devono separare in due classi i punti appartenenti ad un certo insieme si dice *lineare* se una linea (in due dimensioni) o un iperpiano (in *n* dimensioni) possono separare correttamente tutti i punti. In caso contrario, il problema si dice *non lineare*.

Abbiamo visto che un problema lineare può essere risolto usando un perceptron. Infatti, un perceptron con *n* ingressi è in grado di rappresentare un iperpiano *n*-dimensionale. Quindi, un perceptron è in grado di risolvere problemi linearmente separabili in cui gli ingressi devono essere catalogati in due differenti classi separabili tramite una retta (perceptron a due ingressi), un piano (perceptron a tre ingressi), o un iperpiano (perceptron a *n* ingressi).

Un tipico problema non lineare è l'or esclusivo (XOR). Tale operatore, infatti, produce 1 in uscita solo quando uno solo degli ingressi vale 1, altrimenti dà 0. Non esiste alcuna retta che separi i punti (0,1) e (1,0) dai punti (0,0) e (1,1). Per risolvere questo problema si hanno due possibilità:

1) si ricorre a particolari funzioni di uscita non lineari,

2) si usano reti con più strati.

Nel primo approccio si utilizza una rete con *n* ingressi ed un neurone di uscita la cui funzione di uscita è scelta in modo appropriato. Un esempio di funzione di uscita non lineare adatta per i nostri scopi è la seguente:

$$y=(x_1-x_2)^2=\begin{cases}1 \text{ se } x_1\neq x_2\\ 0 \text{ se } x_1=x_2\end{cases}$$

Ovviamente, questo approccio può essere difficile da perseguire qualora la funzione non lineare richiesta sia difficile da individuare.

Nel secondo approccio, si usa una rete con uno o più strati nascosti in modo da modellare due o più rette per separare i dati. Tale rete è detta *multilayer perceptron*.

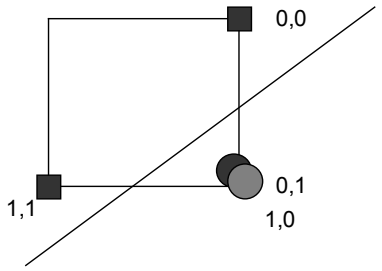
Nel caso dello XOR la rete può essere quella già vista precedentemente e contenente due neuroni nascosti che rappresentano due rette e un neurone di uscita che combina le informazioni prodotte dalle due rette.

Con riferimento alla rete già vista per risolvere lo XOR, consideriamo la seguente tabella che riporta gli ingressi ai neuroni nascosti e le relative uscite.

		ingresso strato nascosto		uscita strato nascosto	
x_1	x_1	neurone 1	neurone 2	neurone 1	neurone 2
1	1	-0.5	-1.5	0	0
1	0	0.5	-0.5	1	0
0	1	0.5	-0.5	1	0
0	0	1.5	0.5	1	1

Possiamo osservare che gli ingressi alla rete risultano trasformati in uscita dallo strato nascosto: il primo livello di pesi (tra lo strato di ingresso e lo strato nascosto) ha spostato il punto originale (0,1) nel punto (1,0). Notiamo anche che (0,0) e (1,1) sono stati scambiati tra loro.

La figura seguente rappresenta l'uscita dallo strato nascosto.

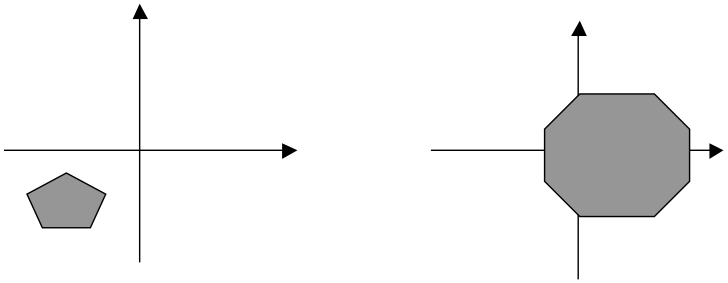


Anche il secondo livello di pesi (che connettono lo strato nascosto con lo strato di uscita) modella una retta. Affinché la rete produca gli output desiderati, occorre quindi che le configurazioni degli input al secondo livello di pesi siano separabili da una retta, come confermato dalla figura precedente.

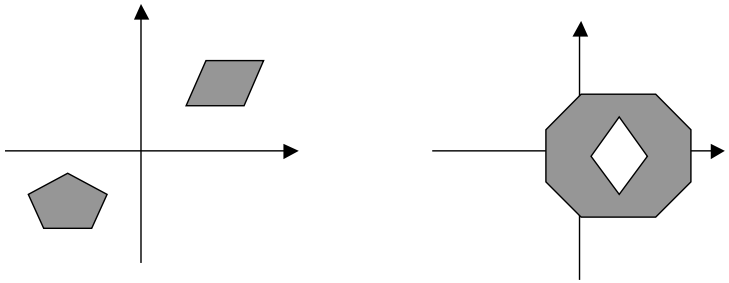
I pesi che definiscono la retta di separazione disegnata nella figura precedente sono quelli già visti quando abbiamo parlato per la prima volta del problema dello XOR.

Osservazioni

Usando reti con uno strato nascosto è possibile formare regioni decisionali convesse nello spazio degli ingressi. Dato che ogni neurone di uno strato separa due regioni, il numero di lati della regione è ≤ al numero di neuroni dello strato nascosto. Ad esempio, possiamo ottenere le seguenti regioni in grigio:



Con due strati nascosti possiamo realizzare regioni decisionali complesse, ad esempio:



La maggior parte dei problemi è risolvibile adottando reti con al massimo due strati nascosti.

Osserviamo che la formazione di regioni decisionali complesse è possibile solo se si adottano funzioni di uscita non lineari. Infatti una rete multistrato con neuroni lineari è equivalente ad una rete con uno strato di ingresso ed uno strato di uscita. Ad esempio, con riferimento ad una rete con uno strato nascosto e neuroni lineari, siano n, p e q il numero di neuroni dei tre strati di ingresso, nascosto e di uscita, rispettivamente. Ricordando che, di solito, la matrice di connessione tra lo strato i e lo strato j si indica con W_{ji}, indichiamo con W_{21} e W_{32} le matrici dei pesi relative alla coppia di strati (ingresso-nascosto) e (nascosto-uscita), rispettivamente. W_{21} e W_{32} hanno dimensioni $p \times n$ e $q \times p$, rispettivamente. L'uscita dallo strato nascosto è data da $Y = W_{21}X$, essendo X il vettore di ingresso. L'uscita dallo strato di uscita è $Z = W_{32}Y = W_{32}W_{21}X$. Definendo $W = W_{32}W_{21}$, otteniamo $Z = WX$, cioè la rete considerata è equivalente ad una rete senza strati nascosti con connessioni espresse da una matrice W, di dimensione $q \times n$, che è il prodotto delle due matrici date.

BACKPROPAGATION

La rete multistrato che abbiamo adottato precedentemente per risolvere il problema dello XOR è stata costruita definendo i pesi appropriati. Ovviamente, ciò che ci interessa trovare è un algoritmo di addestramento supervisionato che permetta alla rete multistrato di trovare da sola i pesi.

Il problema incontrato nell'addestramento delle reti multistrato è il seguente: volendo adottare un meccanismo di aggiornamento dei pesi

simile alla delta rule (in cui l'errore è calcolato come differenza tra l'uscita desiderata e l'uscita effettiva di ciascun neurone) si riesce ad aggiornare solo i pesi relativi ai neuroni di uscita, ma non quelli relativi ai neuroni degli strati nascosti. Infatti, mentre per lo strato di uscita si conosce l'uscita desiderata (tale uscita viene data come secondo elemento delle coppie che costituiscono gli esempi del training set), niente si sa dell'uscita desiderata dai neuroni nascosti.

Questo problema è stato risolto dopo molti anni di disinteresse per le reti neurali (in quanto non si riusciva ad addestrarle) solo nel 1986, quando fu introdotto l'algoritmo di *backpropagation*. Tale algoritmo prevede di calcolare l'errore commesso da un neurone dell'ultimo strato nascosto propagando all'indietro l'errore calcolato sui neuroni di uscita collegati a tale neurone. Lo stesso procedimento è poi ripetuto per tutti i neuroni del penultimo strato nascosto, e così via.

L'algoritmo di backpropagation prevede che, per ogni esempio del training set, i segnali viaggino dall'ingresso verso l'uscita al fine di calcolare la risposta della rete. Dopo di che c'è una seconda fase durante la quale i segnali di errore vengono propagati all'indietro, sulle stesse connessioni su cui nella prima fase hanno viaggiato gli ingressi, ma in senso contrario, dall'uscita verso l'ingresso. Durante questa seconda fase vengono modificati i pesi.

I pesi sono inizializzati con valori casuali. Come funzione di uscita non lineare dei neuroni della rete si adotta in genere la funzione sigmoide (l'algoritmo richiede che la funzione sia derivabile). Tale funzione produce valori tra 0 e 1.

L'algoritmo di backpropagation usa una generalizzazione della delta rule.

Nel seguito useremo *net* per indicare la somma pesata degli ingressi di un neurone.

Possiamo esprimere la derivata dell'errore come segue:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}}$$

Poniamo

$$\delta_j = -\frac{\partial E}{\partial net_j}$$

(osserviamo che questa definizione coincide con la delta rule data nelle pagine precedenti. Lì, infatti, avevamo $\delta_j = -\frac{\partial E}{\partial o_j}$ perché consideravamo neuroni lineari, cioè $o_j = net_j$).

Riscriviamo l'equazione precedente:

$$\delta_j = -\frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j}$$

Poiché l'errore commesso sul *k*-esimo esempio del training set è:

$$E_k = \frac{1}{2}\sum_j (t_j - o_j)^2$$

abbiamo

$$\frac{\partial E}{\partial o_j} = -(t_j - o_j)$$

Per la funzione di attivazione *f* (di solito, la funzione logistica) l'uscita è:

$$o_j = f(net_j)$$

e quindi

$$\frac{\partial o_j}{\partial net_j} = f'(net_j)$$

da cui

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

Essendo

$$net_j = \sum_i x_i w_{ij}$$

si ha

$$\frac{\partial net_j}{\partial w_{ij}} = x_i$$

per cui, tornando alla formula di partenza risulta

$$\frac{\partial E}{\partial w_{ij}} = -(t_j - o_j) f'(net_j) x_i = -\delta_j x_i$$

Quindi, applicando il learning rate, abbiamo la seguente formula per la modifica dei pesi, sulla base della delta rule generalizzata:

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

L'errore δ_j è calcolabile per un neurone di uscita, ma non per un neurone nascosto perché, come già detto, non conosciamo la sua uscita desiderata. Comunque, un neurone nascosto può essere adattato in modo proporzionale al suo contributo all'errore sullo strato successivo (verso l'uscita della rete). Fissando l'attenzione su un neurone dell'ultimo strato nascosto, possiamo dire che l'errore commesso da tale neurone può essere calcolato come somma degli errori commessi da tutti i neuroni di uscita collegati a tale neurone nascosto. Il contributo di ciascuno di tali errori dipende, ovviamente, sia dalla dimensione dell'errore commesso dal relativo neurone di uscita sia dal peso sulla connessione tra il neurone nascosto e il neurone di uscita. In altri termini, un neurone di uscita con un grosso errore contribuisce in maniera notevole all'errore di ogni neurone nascosto a cui è connesso con un peso elevato. Per un neurone nascosto l'errore è dato da:

$$\delta_j = f'(net_j) \sum_s \delta_s w_{js}$$

dove *s* è l'indice dei neuroni dello strato che trasmette all'indietro l'errore.

Come detto, una funzione spesso usata è la funzione logistica

$$f(net_j) = \frac{1}{1 + \exp(-net_j)}$$

La derivata di tale funzione è:

$$f'(net_j) = \frac{\exp(-net_j)}{(1 + \exp(-net_j))^2}$$

$$= \frac{1}{1 + \exp(-net_j)} \left(1 - \frac{1}{1 + \exp(-net_j)} \right)$$

$$= f(net_j)[1 - f(net_j)]$$

Osservazione

Abbiamo già osservato, parlando del perceptron, che il learning rate η non deve avere valori troppo bassi né troppo alti per evitare, rispettivamente, tempi di addestramento troppo lunghi od oscillazioni dell'errore. Esistono due tecniche per risolvere questo problema. La prima è la stessa vista per il perceptron e prevede di variare η nel tempo. La seconda prevede di ridurre la probabilità di oscillazione dei pesi usando un termine α, detto *momentum*, che è una costante di proporzionalità (compresa tra 0 e 1) alla precedente variazione dei pesi. La legge di apprendimento diventa quindi:

$$\Delta w_{ij}(n+1) = \eta \delta_j o_i + \alpha \Delta w_{ij}(n).$$

In questo modo, il cambiamento dei pesi per l'esempio *n*+1 dipende dal cambiamento apportato ai pesi per l'esempio *n*.

Algoritmo di backpropagation

Dato un training set costituito da *m* esempi (X_k, T_k), l'addestramento di una rete multistrato, con funzioni di trasferimento sigmoidi, avviene tramite i seguenti passi:

1. si inizializzano i pesi con valori casuali (in genere, con valori non troppo elevati);
2. si presenta un ingresso X_k e si calcolano le uscite $o_j = \frac{1}{1 + \exp(-net_j)}$ di tutti i neuroni della rete;

3. dato T_k, si calcolano l'errore δ_j e la variazione dei pesi Δw_{ij} per ogni neurone dello strato di uscita:

   $$\delta_j = (t_j - o_j) f'(net_j) = (t_j - o_j) o_j (1 - o_j)$$

4. partendo dall'ultimo strato nascosto e procedendo all'indietro, calcolare

   $$\delta_j = f'(net_j) \sum_s \delta_s w_{js} = o_j (1 - o_j) \sum_s \delta_s w_{js}$$

5. per tutti gli strati aggiornare i pesi:

   $$\Delta w_{ij}(n+1) = \eta \delta_j o_i + \alpha \Delta w_{ij}(n)$$

6. ripetere il processo dal punto 2 finchè non si siano presentati tutti gli *m* esempi del training set;
7. si calcola l'errore medio sul training set (oppure l'errore globale); se l'errore è al di sotto di una soglia prefissata, l'algoritmo termina (si è raggiunta la convergenza), altrimenti si ripete un intero ciclo di presentazione del training set.

Un ciclo di presentazione degli esempi del training set è detto *epoca*.

Esistono due modalità di applicazione dell'algoritmo di backpropagation: nella modalità *batch* i pesi sono aggiornati dopo aver presentato alla rete tutti gli esempi del training set, nella modalità *on-line* (o *incrementale*) i pesi sono aggiornati dopo la presentazione di ogni esempio. Nell'algoritmo precedente si è fatto riferimento a quest'ultima modalità.