INTELLIGENZA ARTIFICIALE

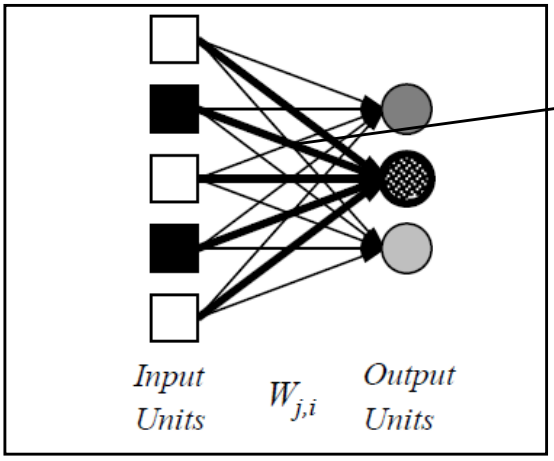
ANNO ACCADEMICO
2008-2009

Machine Learning:Reti Neurali

Sommario

- Apprendimento nel percettrone a sigmoide
- Reti feed-forward multistrato
- Apprendimento nel percettrone multistrato: back-propagation
- Conclusioni

Apprendimento nel percettrone



Sinapsi
$W_{i,j}$

Cosa significa apprendere per un percettrone?

Ricerca della ipotesi h_W

Come sappiamo, apprendere significa ricercare la migliore ipotesi h_W nello spazio delle ipotesi H che più si avvicina alla ipotesi target h^*

↓

Una ipotesi generica h viene individuata non appena si fissino le sinapsi della rete W_j : W->h

↓

Apprendere significa quindi modificare le sinapsi in modo opportuno

↓

Apprendere=Minimizzare una funzione costo E

Prerequisiti matematici

Prerequisiti matematici

E' importante al fine di comprendere gli algoritmi di apprendimento, ricordare alcuni passaggi matematici:

- Derivata parziale di una funzione composta
- Vettore gradiente di una funzione reale a più variabili reali
- Metodo della discesa del gradiente come ricerca di un minimo

Rivediamo rapidamente questi concetti

Derivata parziale

In analisi matematica, la derivata parziale è una prima generalizzazione del concetto di derivata di una funzione reale alle funzioni reali di più variabili

Definizione Derivata parziale

Sia $E \subseteq \mathbb{R}^n$ aperto. Consideriamo una funzione di n variabili $f(\vec{x}) = f(x_1, x_2, \ldots, x_n) \ : \ E \to \mathbb{R}$. Si definisce derivata parziale di f in $\vec{x} \in E$, rispetto alla variabile k-esima x_k, il limite, se esiste finito,

$$\frac{\partial f(\vec{x})}{\partial x_k} = \lim_{h \to 0} \frac{f(x_1, x_2, \ldots, x_k + h, \ldots, x_n) - f(x_1, x_2, \ldots, x_n)}{h}$$

Tale limite è a volte chiamato limite del rapporto incrementale di f nel punto $\vec{x}$ rispetto alla variabile x_k.

Derivate parziali in $\mathbb{R}^2$ [modifica]

Per chiarire le idee, consideriamo l'esempio più semplice, cioè una funzione f con dominio in $\mathbb{R}^2$, insieme formato da tutte le coppie ordinate (x, y) con $x, y \in \mathbb{R}$, e con valori in $\mathbb{R}$. Tale funzione in ogni punto (x_0, y_0) del proprio dominio può avere due derivate parziali:

- derivata parziale di f rispetto a x:

$$f_x(x_0, y_0) = \frac{\partial f}{\partial x}(x_0, y_0) = \lim_{h \to 0} \frac{f(x_o + h, y_0) - f(x_0, y_0)}{h}$$

- derivata parziale di f rispetto a y:

$$f_y(x_0, y_0) = \frac{\partial f}{\partial y}(x_0, y_0) = \lim_{k \to 0} \frac{f(x_o, y_0 + k) - f(x_0, y_0)}{k}$$

Se entrambi i limiti esistono finiti, allora la funzione f si dice derivabile in $(x_0, y_0) \in \mathbb{R}^2$. Il vettore che ha per componenti f_x e f_y è detto gradiente della funzione f in (x_0, y_0)

http://it.wikipedia.org/wiki/Derivata_parziale

Vettore gradiente

In matematica il gradiente di un campo scalare è una funzione a valori reali di più variabili reali, quindi definita in una regione di uno spazio a due, tre o più dimensioni. Il gradiente di una funzione è definito come il vettore che ha per componenti cartesiane le derivate parziali della funzione.
Il gradiente rappresenta quindi la direzione di massimo incremento di una funzione reale di variabili reali

$$\operatorname{grad} f(x_0, y_0) = \nabla f(x_0, y_0) = \begin{pmatrix} f_x(x_0, y_0) \\ f_y(x_0, y_0) \end{pmatrix}$$
$$= \mathbf{i} \frac{\partial f(x, y)}{\partial x} + \mathbf{j} \frac{\partial f(x, y)}{\partial y}$$

In $\mathbb{R}^3$ si definisce similmente:

$$\operatorname{grad} f(x_0, y_0, z_0) = \nabla f(x_0, y_0, z_0) = \begin{pmatrix} f_x(x_0, y_0, z_0) \\ f_y(x_0, y_0, z_0) \\ f_z(x_0, y_0, z_0) \end{pmatrix}$$
$$= \mathbf{e}_x \frac{\partial f(x, y, z)}{\partial x} + \mathbf{e}_y \frac{\partial f(x, y, z)}{\partial y} + \mathbf{e}_z \frac{\partial f(x, y, z)}{\partial z}$$

Discesa del gradiente

La discesa del gradiente è una tecnica di ottimizzazione di tipo locale. Data una funzione matematica multidimensionale, la discesa del gradiente consente di trovare un minimo locale di questa funzione

La discesa di gradiente d'errore trova solo minimi locali di funzione. Puó peró anche esser utilizzata nella ricerca di un mimimo globale, scegliendo a caso un nuovo punto iniziale una volta che si sia trovato un minimo locale, e ripetendo l'operazione moltissime volte (simulated annealing).

Apprendimento

Caso I
Apprendimento nel percettrone a sigmoide

• **Obiettivo**: calibrare i pesi (sinapsi) della rete in modo tale da **minimizzare** una funzione di costo ***E***
• **Apprendimento**=ricerca di ottimizzazione nello spazio dei pesi. Equivale alla ricerca della migliore ipotesi nello spazio delle ipotesi H
• Misura di errore "classica": **E=somma dei quadrati degli errori**
•Tipo di addestramento: **supervisionato** (coppie di addestramento <x,f(x)> note)
• **Errore**= differenza tra l'output desiderato *y* e l'output della rete $h_W(x)$

$$E = \frac{1}{2}Err^2 \equiv \frac{1}{2}(y - h_W(\vec{x}))^2$$

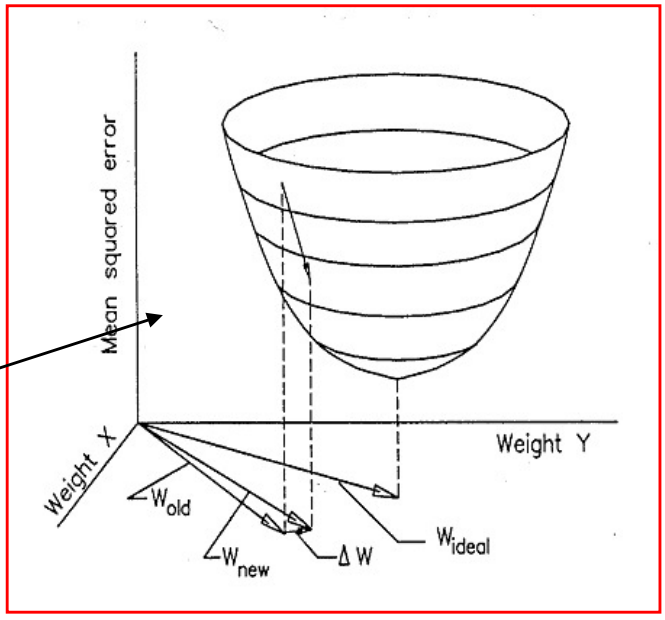
• essendo: $\vec{x}$ il vettore di input, $h_W(\vec{x})$ il valore della funzione di output (ipotesi da determinare) e y il valore di output corrispondente ad $\vec{x}$

Metodo della discesa del gradiente (delta rule)

L'apprendimento consiste nel minimizzare la funzione di costo E, modificando i pesi nella direzione opposta al gradiente della funzione stessa (*discesa del gradiente*) alla ricerca del minimo

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

Ci si sposta alla ricerca del minimo locale



Significato geometrico del metodo della discesa del gradiente

Metodo della discesa del gradiente

$$\frac{\partial E}{\partial W_j} = Err \times \frac{\partial Err}{\partial W_j} = Err \times \frac{\partial}{\partial W_j}(y - g(\sum_{j=0}^{n} W_j x_j))$$
$$= -Err \times g'(in) \times x_j$$

function APPRENDIMENTO-PERCETTRONE(*esempi*,*rete*) **returns** un'ipotesi percettrone
inputs: *esempi*, un insieme di esempi, ognuno con input $\mathbf{x} = x_1, \ldots, x_n$ e output y
rete, un percettrone con pesi W_j, j=0…n e funzione di attivazione g

repeat
for each e **in** *esempi* **do**
$in \longleftarrow \sum_{j=0}^{n} W_j x_j[e]$
$Err \longleftarrow y[e] - g(in)$
$W_j \longleftarrow W_j + \alpha \times Err \times g'(in) \times x_j[e]$
until non è soddisfatto un qualche criterio di terminazione
return IPOTESI-RETE-NEURALE (*rete*)

α = *tasso di apprendimento*. Esprime la velocità di spostamento sulla curva alla ricerca del minimo

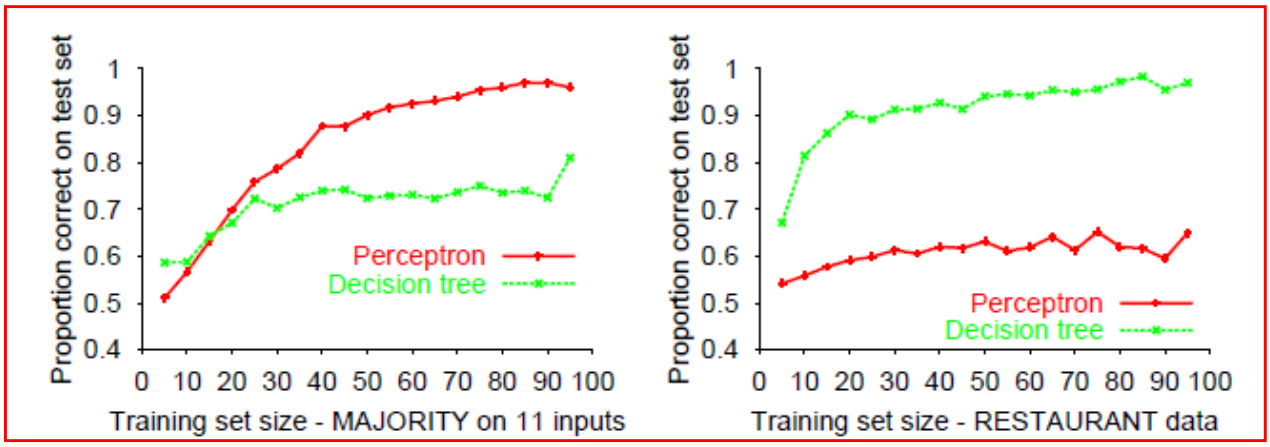
Considerazioni

- Si presume che la funzione *g* sia derivabile
- Per percettroni a soglia, il fattore *g'(in)* è omesso e l'algoritmo di aggiornamento dei pesi diviene: $W_j = W_j + \alpha \times Err \times x_j$
- Per percettroni a sigmoide si ha: *g'=g(1-g)* facilmente trattabile
- Gli esempi di addestramento vengono fatti passare attraverso la rete uno per volta
- Ogni ciclo attraverso tutti gli esempi prende il nome di epoca
- Le epoche sono ripetute secondo un ben preciso criterio di terminazione
- Il cambiamento dei pesi è dato dalla differenza tra target e output moltiplicata per l'attività presinaptica

La regola delta è plausibile dal punto di vista psico-biologico e corrisponde formalmente alla regola di Rescorla-Wagner del condizionamento classico.

Rescorla, R. A., and Wagner, A. R. (1972) *A theory of Pavlovian conditioning: Variations in the effectiveness of reinforcement and nonreinforcement*, Classical Conditioning II, A. H. Black and W. F. Prokasy, Eds., pp. 64-99. Appleton-Century-Crofts.

Prestazioni



• Il percettrone lavora bene su funzioni linearmente separabili (funzione di maggioranza)
• L'albero di decisione lavora meglio su funzione non linearmente separabile (ristorante)

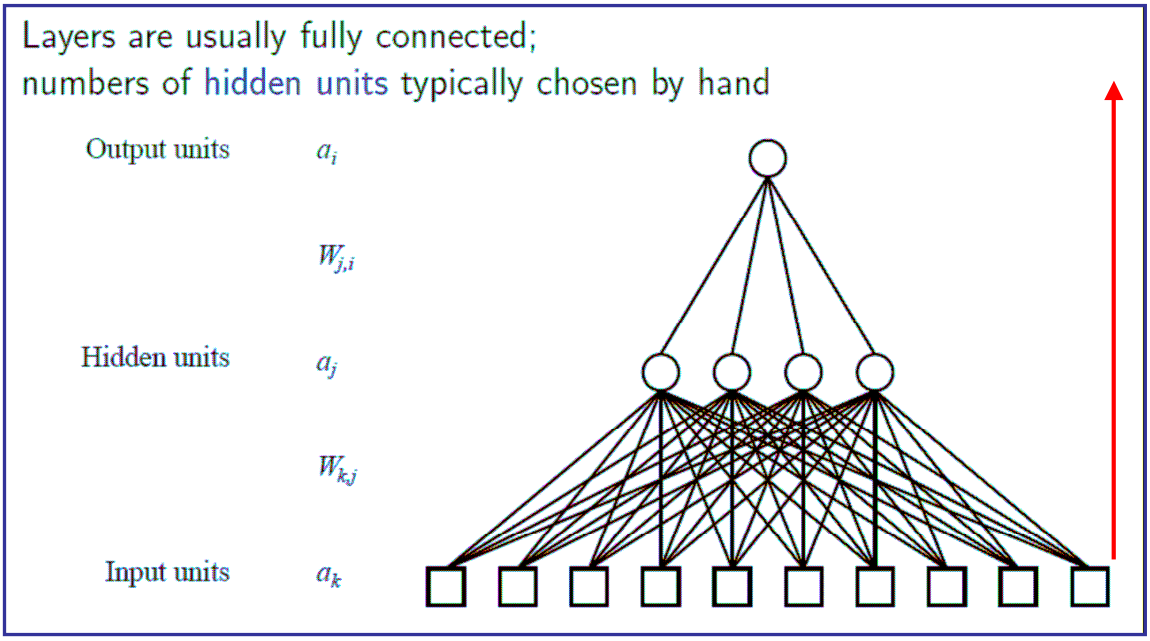
Reti neurali feed-forward multistrato

• Reti in cui esiste un verso di propagazione del segnale dall'input all'output
• Ciascun nodo dello strato *i-mo* è collegato con tutti i nodi dello strato *i+1-mo*
• Percettrone multistrato

Esempio del ristorante
10 neuroni di input
1 strato nascosto 4 neuroni
1 neurone di output

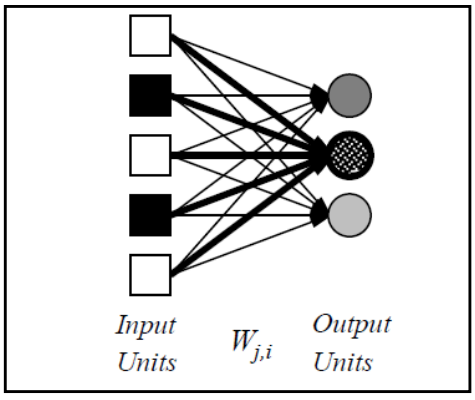
Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	$$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	$	F	F	Burger	30–60	T

Rete neurale per il problema ristorante

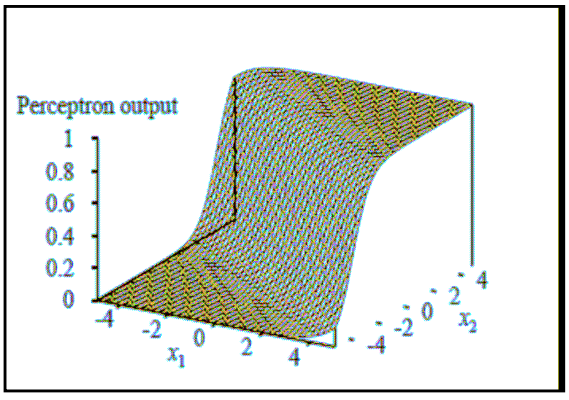


L'aggiunta dello strato nascosto consente di ampliare lo spazio delle ipotesi H

Spazio delle ipotesi H per il percettrone semplice (a soglia morbida)



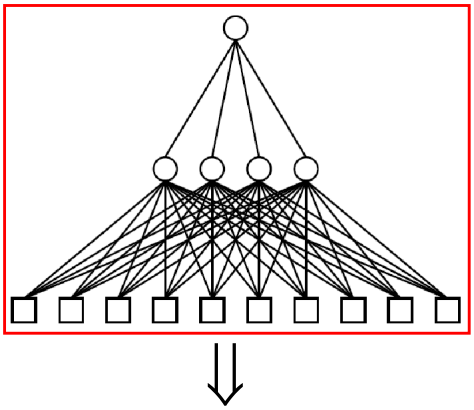
=>



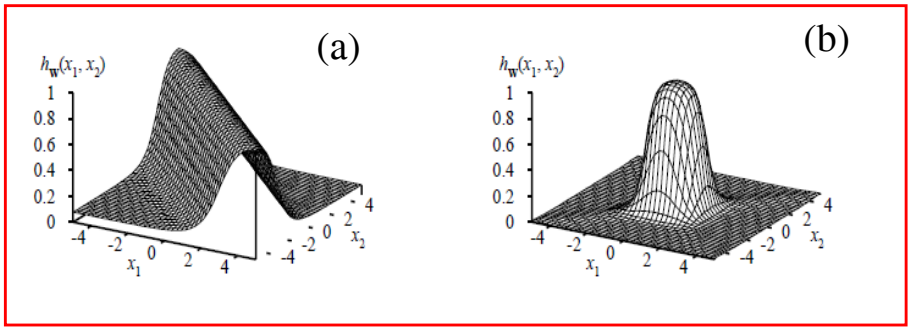
Ciascuna unità *a* suddivide lo spazio H in due parti secondo una sigmoide

Spazio delle ipotesi H per il percettrone multistrato

• Fig. (a): cresta prodotta da due funzioni a soglia morbida
• Fig. (b): protuberanza prodotta dalla combinazione di due creste



Migliore rappresentazione dello spazio H

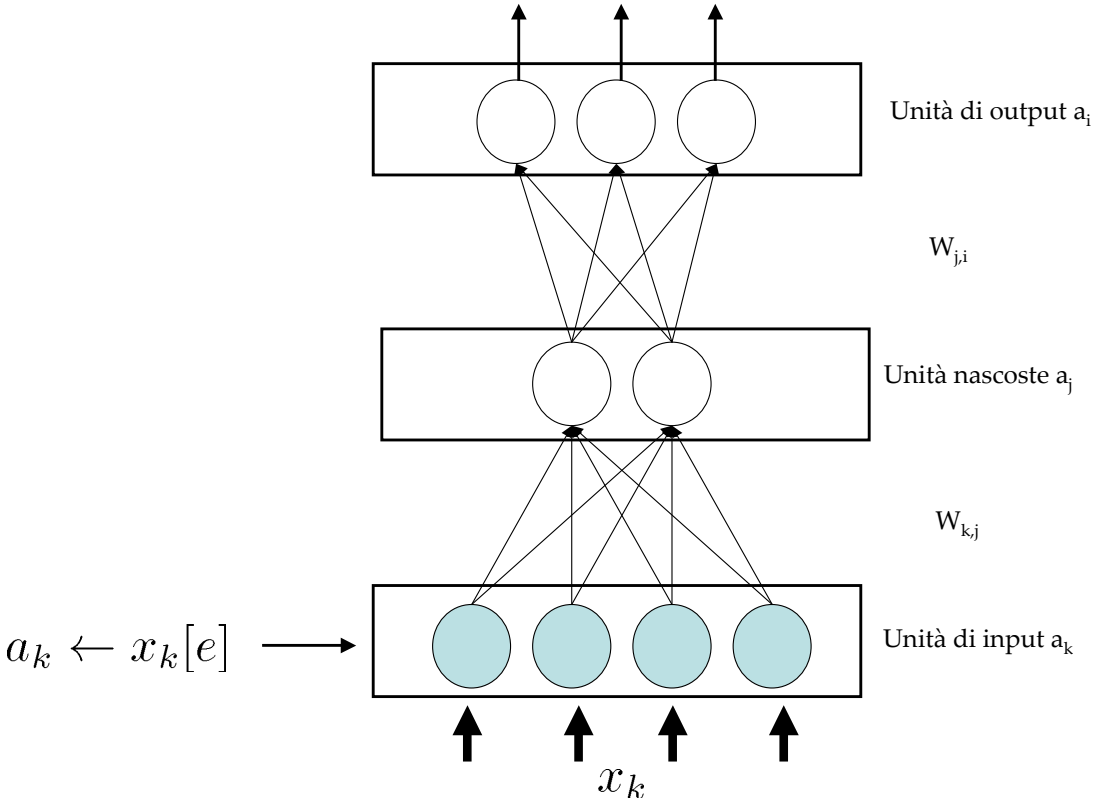


Reti neurali feed-forward multistrato: Apprendimento

• Algoritmo di apprendimento: Back-Propagation
• Si può avere vettore di output $h_W(x)$ invece che un solo valore
• L'algoritmo utilizza un metodo di retropropagazione dell'errore Err= y- $h_W(x)$ agli strati intermedi
• La propagazione all'indietro dell'errore rende l'algoritmo implausibile dal punto di vista biologico però funziona (☺)
• **Pro**
  • efficienza computazionale
  • permette di addestrare reti con un qualsiasi numero di strati nascosti
• **Contro**
  • problema dei minimi locali
  • discesa del gradiente lenta

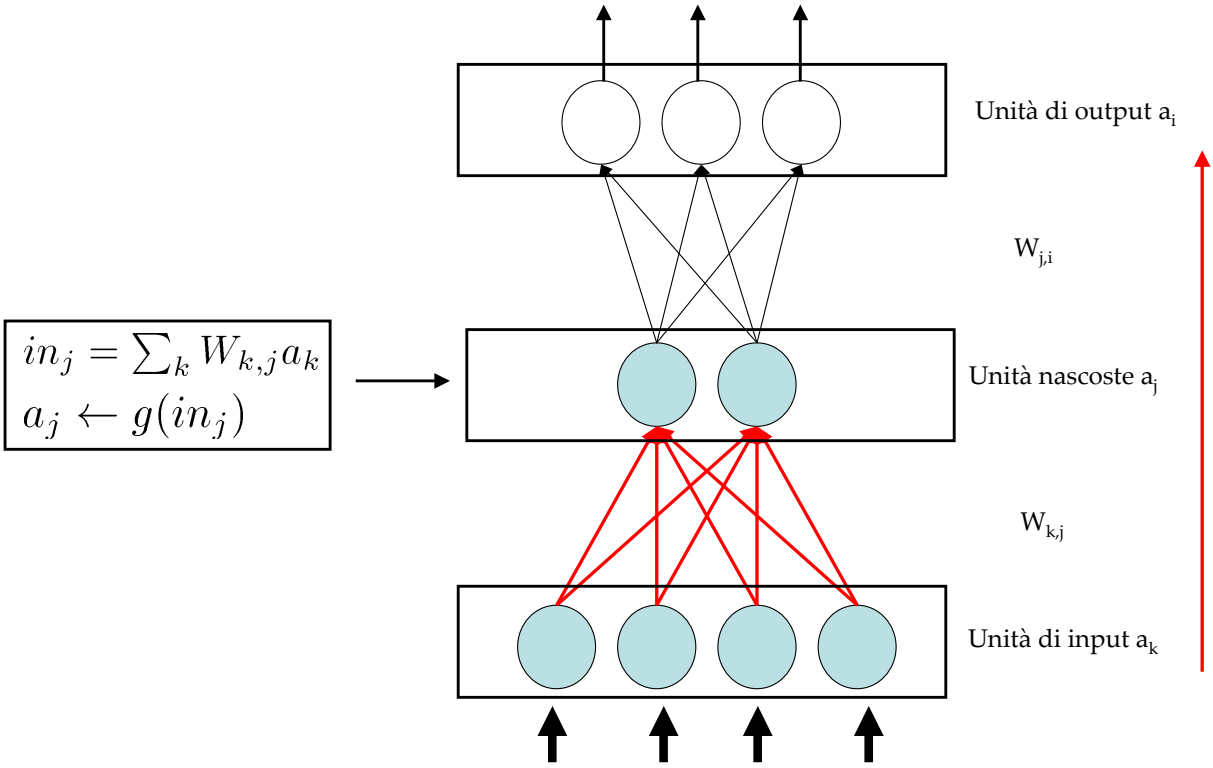
Back Propagation

1. Presentazione pattern d'ingresso



Back Propagation

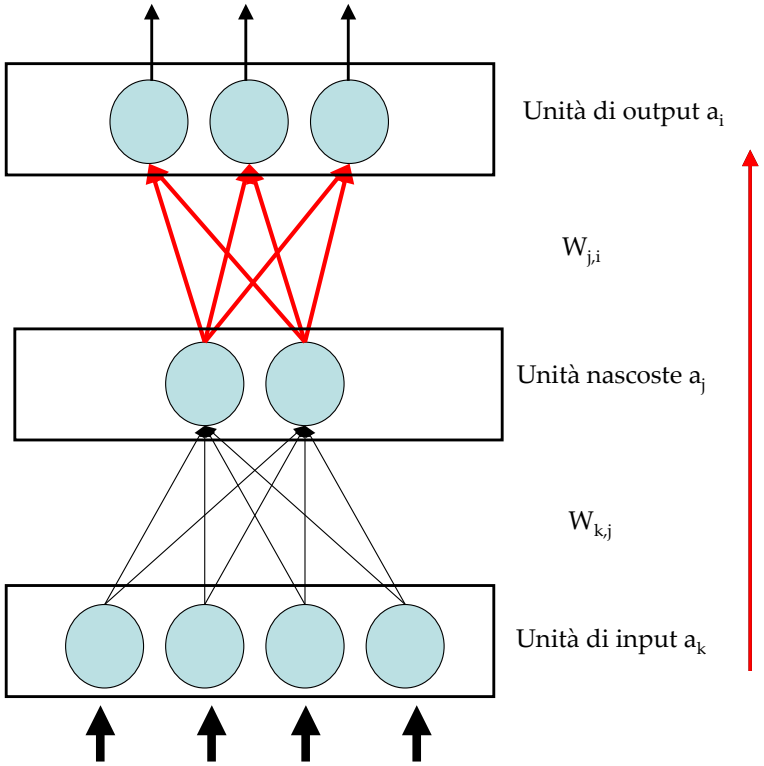
2. Propagazione dell'input in avanti sullo strato nascosto



Back Propagation

3. Propagazione dallo strato nascosto allo strato di output

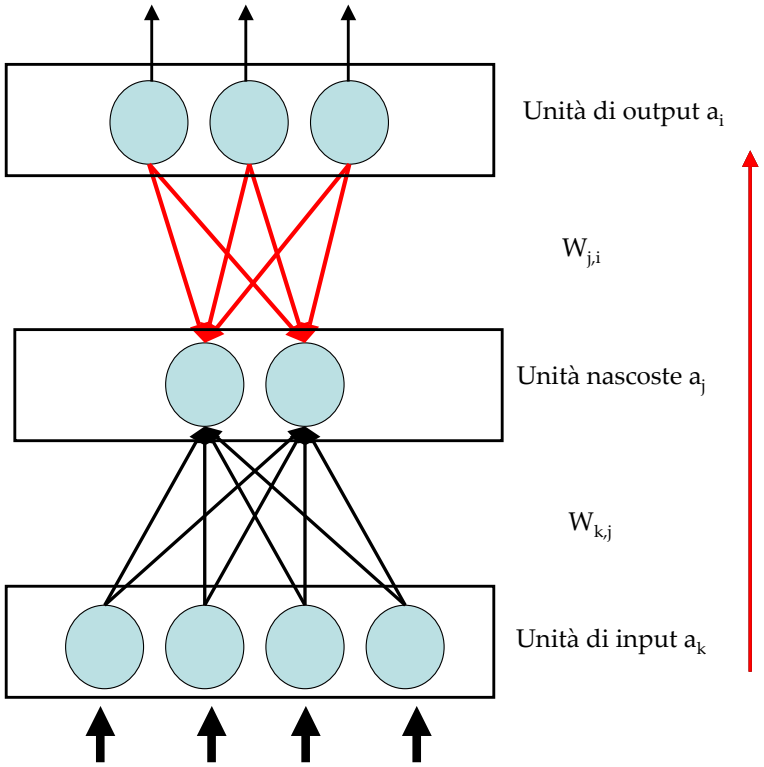
$$\Delta_i \leftarrow g'(in_i) \times (y_i[e] - a_i) \longrightarrow$$



Back Propagation

4. Retropropagazione dell'errore

$$\Delta_j \leftarrow g'(in_j) \times \sum_i W_{j,i}\Delta_i \longrightarrow$$



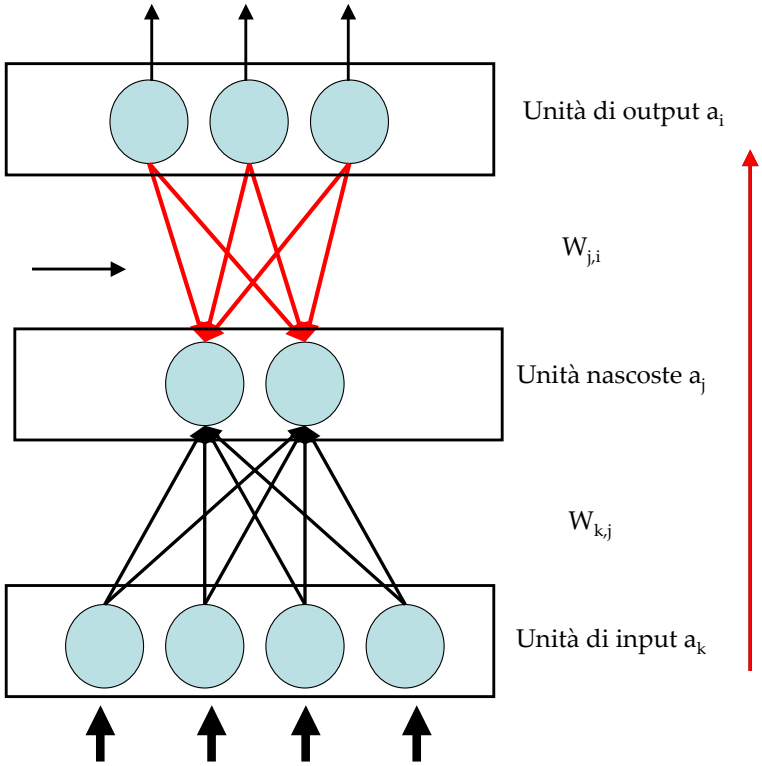
L'errore si **retropropaga** su ciascun nodo proporzionalmente

alla forza di connessione tra il nodo nascosto e il nodo di output

Back Propagation

5. Aggiornamento dei pesi

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$$



Back-propagation learning

Output layer: same as for single-layer perceptron,

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$$

where $\Delta_i = Err_i \times g'(in_i)$

Hidden layer: back-propagate the error from the output layer:

$$\Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i \ .$$

Update rule for weights in hidden layer:

$$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j \ .$$

(Most neuroscientists deny that back-propagation occurs in the brain)

Algoritmo BP in pseudolinguaggio

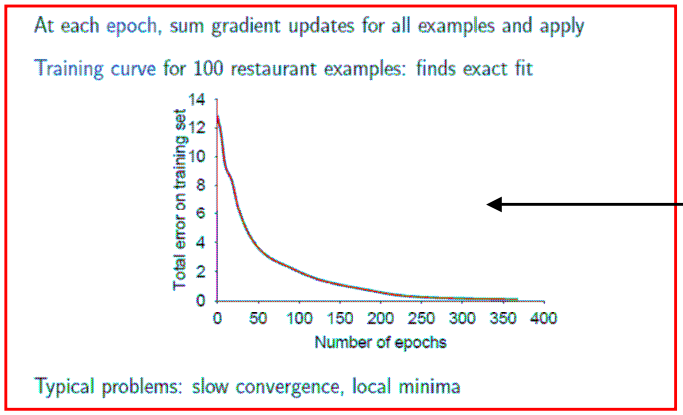
function APPRENDIMENTO-PROP-INDIETRO(*esempi*,*rete*) **returns** una rete neurale
inputs: *esempi*, un insieme di esempi, ognuno con vettore di input **x** e vettore di output **y**
rete, una rete feed-forward multistrato con L strati, pesi $W_{i,i}$ e funzione di attivazione g

repeat
for each e **in** *esempi* **do**
for each nodo k nello strato di input **do** $a_k \leftarrow x_k[e]$
for l=2 **to** L **do**
$in_j \leftarrow \sum_k W_{j,k} a_k$
$a_j \leftarrow g(in_j)$
for each nodo i nello strato di output **do**
$\Delta_i \leftarrow g'(in_i) \times (y_i[e] - a_i)$
for l = L-1 **to** 1 **do**
for each nodo j nello strato l **do**
$\Delta_j \leftarrow g'(in_j) \times \sum_i W_{j,i} \Delta_i$
for each nodo i nello strato l+1 **do**
$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$
until non è soddisfatto un qualche criterio di terminazione
return IPOTESI-RETE-NEURALE (*rete*)

Calcolo del gradiente

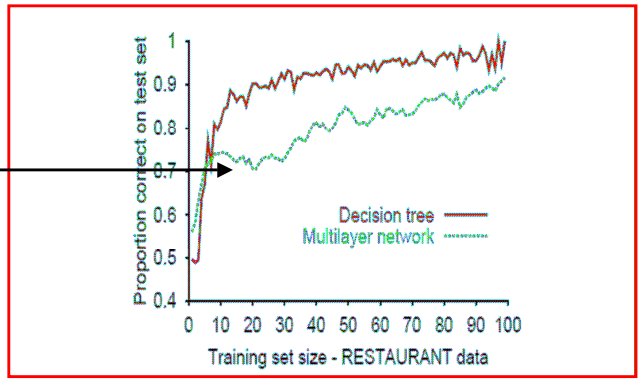
$$
\begin{aligned}
\frac{\partial E}{\partial W_{k,j}} &= -\sum_i (y_i - a_i) \frac{\partial a_i}{\partial W_{k,j}} = -\sum_i (y_i - a_i) \frac{\partial g(in_i)}{\partial W_{k,j}} \\
&= -\sum_i (y_i - a_i) g'(in_i) \frac{\partial in_i}{\partial W_{k,j}} = -\sum_i \Delta_i \frac{\partial}{\partial W_{k,j}} \left(\sum_j W_{j,i} a_j \right) \\
&= -\sum_i \Delta_i W_{j,i} \frac{\partial a_j}{\partial W_{k,j}} = -\sum_i \Delta_i W_{j,i} \frac{\partial g(in_j)}{\partial W_{k,j}} \\
&= -\sum_i \Delta_i W_{j,i} g'(in_j) \frac{\partial in_j}{\partial W_{k,j}} \\
&= -\sum_i \Delta_i W_{j,i} g'(in_j) \frac{\partial}{\partial W_{k,j}} \left(\sum_k W_{k,j} a_k \right) \\
&= -\sum_i \Delta_i W_{j,i} g'(in_j) a_k = -a_k \Delta_j
\end{aligned}
$$

Curve di addestramento di BP per l'esempio del ristorante



Curva dell'errore

Curva delle prestazioni sul test set



Conclusioni

- Apprendimento come ricerca del minimo locale di una funzione costo E
- Apprendimento nel percettrone semplice
  - delta rule
- Apprendimento nel percettrone multistrato
  - backpropagation