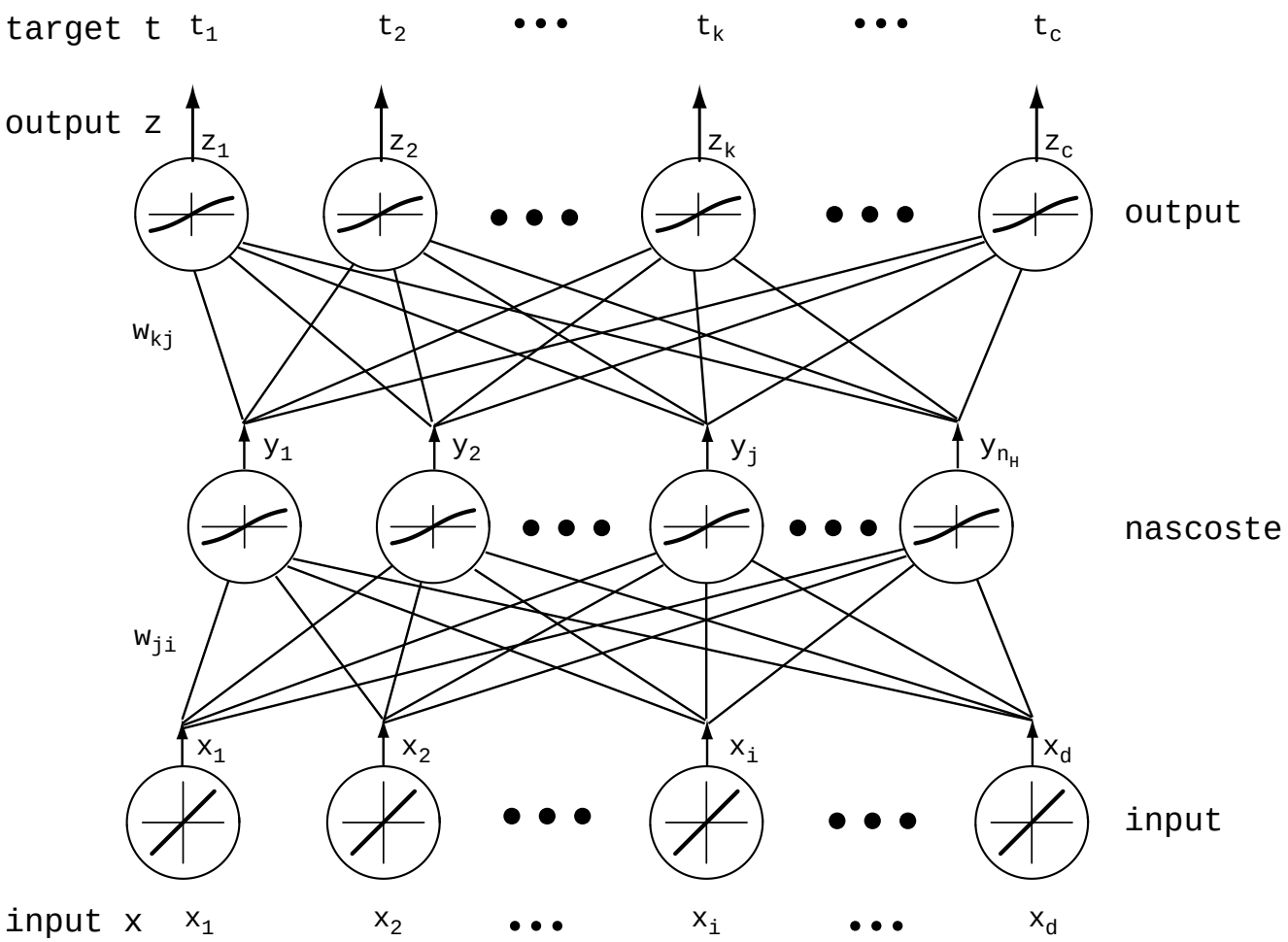
Reti Neurali Feed-forward: notazione



Reti Neurali Feed-forward: notazione

- d unità di ingresso, dimensione dei dati in ingresso $\vec{x} \equiv (x_1, \ldots, x_d)$
  ($d+1$ se si include la soglia nel vettore dei pesi $\vec{x}' \equiv (x_0, x_1, \ldots, x_d)$)
- N_H unità nascoste (con output $\vec{y} \equiv (y_1, \ldots, y_{N_H})$)
- c unità di output, dimensione dei dati in output $\vec{z} \equiv (z_1, \ldots, z_c)$
- c, dimensione dei dati desiderati $\vec{t} \equiv (t_1, \ldots, t_c)$
- w_{ji} peso dalla unità di ingresso i alla unità nascosta j
- w_{kj} peso dalla unità nascosta j alla unità di output k

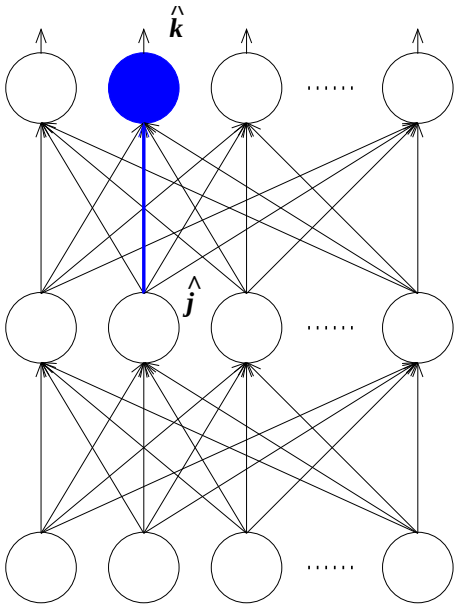
La funzione errore, considerando che si hanno c unità di output, diventa

$$E[\vec{w}] = \frac{1}{2cN_{Tr}} \sum_{(\vec{x}^{(p)}, \vec{t}^{(p)}) \in Tr} \sum_{k=1}^{c} \left(t_k^{(p)} - z_k(\vec{x}^{(p)}) \right)^2$$

Calcolo del gradiente per i pesi di una unità di output

Fissiamo gli indici $\hat{k}$ e $\hat{j}$:

$$\frac{\partial E}{\partial w_{\hat{k}\hat{j}}} = \frac{\partial}{\partial w_{\hat{k}\hat{j}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

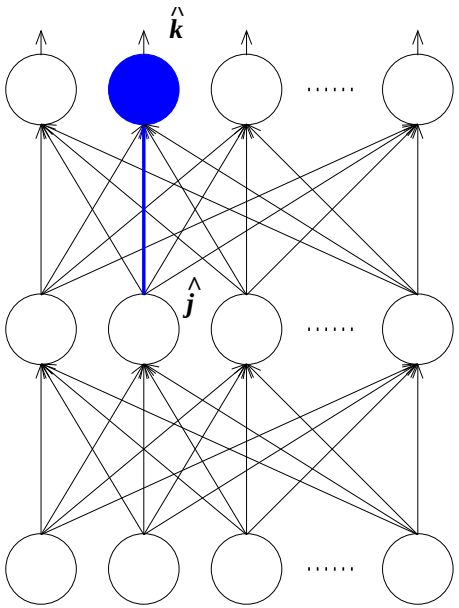


Calcolo del gradiente per i pesi di una unità di output

Fissiamo gli indici $\hat{k}$ e $\hat{j}$:

$$\frac{\partial E}{\partial w_{\hat{k}\hat{j}}} = \frac{\partial}{\partial w_{\hat{k}\hat{j}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \frac{\partial}{\partial w_{\hat{k}\hat{j}}} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$



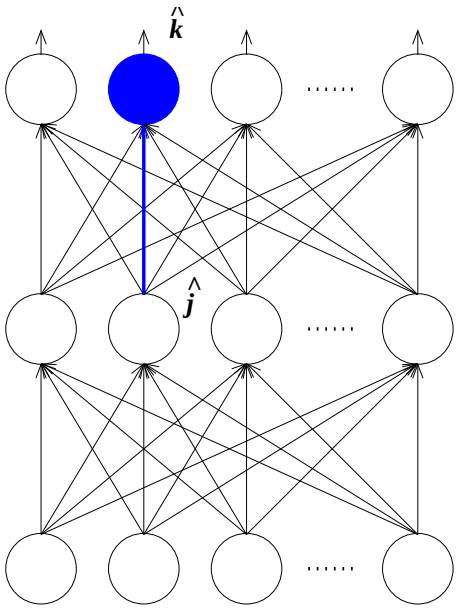
Calcolo del gradiente per i pesi di una unità di output

Fissiamo gli indici $\hat{k}$ e $\hat{j}$:

$$\frac{\partial E}{\partial w_{\hat{k}\hat{j}}} = \frac{\partial}{\partial w_{\hat{k}\hat{j}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \frac{\partial}{\partial w_{\hat{k}\hat{j}}} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} 2(t_{\hat{k}}^{(p)} - z_{\hat{k}}^{(p)}) \frac{\partial}{\partial w_{\hat{k}\hat{j}}} (t_{\hat{k}}^{(p)} - z_{\hat{k}}^{(p)})$$



Calcolo del gradiente per i pesi di una unità di output

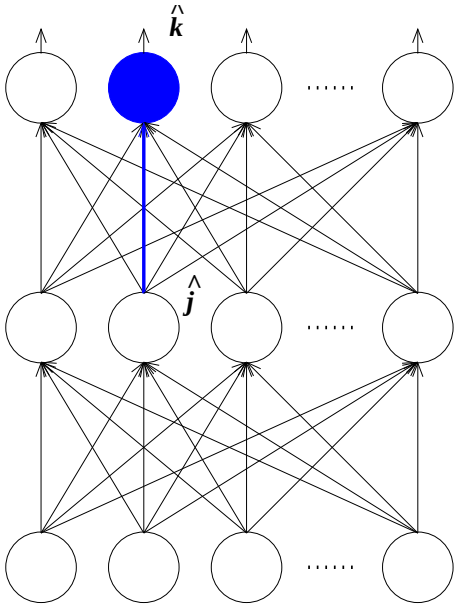
Fissiamo gli indici $\hat{k}$ e $\hat{j}$:

$$\frac{\partial E}{\partial w_{\hat{k}\hat{j}}} = \frac{\partial}{\partial w_{\hat{k}\hat{j}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \frac{\partial}{\partial w_{\hat{k}\hat{j}}} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} 2(t_{\hat{k}}^{(p)} - z_{\hat{k}}^{(p)}) \frac{\partial}{\partial w_{\hat{k}\hat{j}}} (t_{\hat{k}}^{(p)} - z_{\hat{k}}^{(p)})$$

$$= \frac{1}{cN_{Tr}} \sum_{p \in Tr} (t_{\hat{k}}^{(p)} - z_{\hat{k}}^{(p)}) \frac{\partial}{\partial w_{\hat{k}\hat{j}}} (t_{\hat{k}}^{(p)} - \sigma(\vec{w}_{\hat{k}} \cdot \vec{y}^{(p)}))$$



Calcolo del gradiente per i pesi di una unità di output

Fissiamo gli indici $\hat{k}$ e $\hat{j}$:

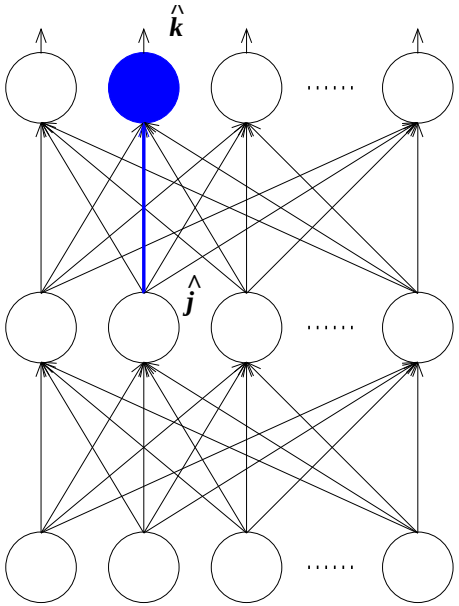
$$\frac{\partial E}{\partial w_{\hat{k}\hat{j}}} = \frac{\partial}{\partial w_{\hat{k}\hat{j}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \frac{\partial}{\partial w_{\hat{k}\hat{j}}} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} 2(t_{\hat{k}}^{(p)} - z_{\hat{k}}^{(p)}) \frac{\partial}{\partial w_{\hat{k}\hat{j}}} (t_{\hat{k}}^{(p)} - z_{\hat{k}}^{(p)})$$

$$= \frac{1}{cN_{Tr}} \sum_{p \in Tr} (t_{\hat{k}}^{(p)} - z_{\hat{k}}^{(p)}) \frac{\partial}{\partial w_{\hat{k}\hat{j}}} (t_{\hat{k}}^{(p)} - \sigma(\vec{w}_{\hat{k}} \cdot \vec{y}^{(p)}))$$

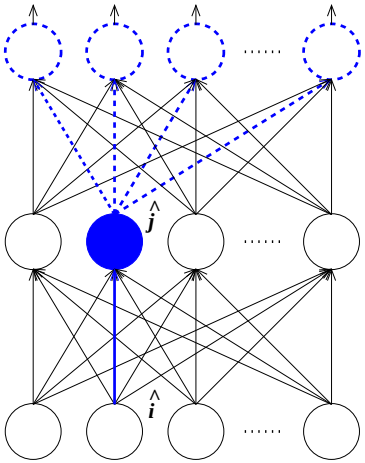
$$= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} (t_{\hat{k}}^{(p)} - z_{\hat{k}}^{(p)}) \sigma'(\vec{w}_{\hat{k}} \cdot \vec{y}^{(p)}) y_{\hat{j}}^{(p)}$$



Calcolo del gradiente per i pesi di una unità nascosta

Fissiamo gli indici $\hat{j}$ e $\hat{i}$:

$$\frac{\partial E}{\partial w_{\hat{j}\hat{i}}} = \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

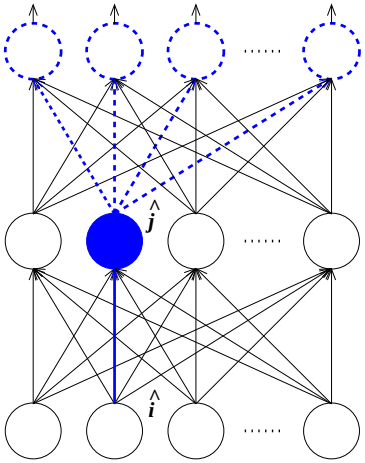


Calcolo del gradiente per i pesi di una unità nascosta

Fissiamo gli indici $\hat{j}$ e $\hat{i}$:

$$\frac{\partial E}{\partial w_{\hat{j}\hat{i}}} = \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (t_k^{(p)} - z_k^{(p)})^2$$



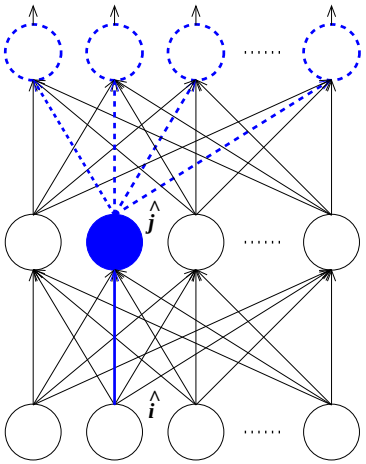
Calcolo del gradiente per i pesi di una unità nascosta

Fissiamo gli indici $\hat{j}$ e $\hat{i}$:

$$\frac{\partial E}{\partial w_{\hat{j}\hat{i}}} = \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (t_k^{(p)} - z_k^{(p)})^2$$

$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} 2(t_k^{(p)} - z_k^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (-z_k^{(p)})$$



Calcolo del gradiente per i pesi di una unità nascosta

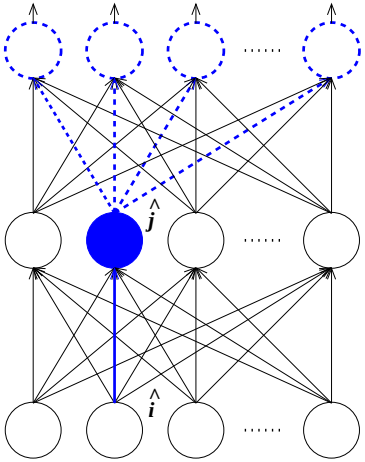
Fissiamo gli indici $\hat{j}$ e $\hat{i}$:

$$\frac{\partial E}{\partial w_{\hat{j}\hat{i}}} = \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (t_k^{(p)} - z_k^{(p)})^2$$

$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} 2(t_k^{(p)} - z_k^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (-z_k^{(p)})$$

$$= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \vec{w}_k \cdot \vec{y}^{(p)}$$

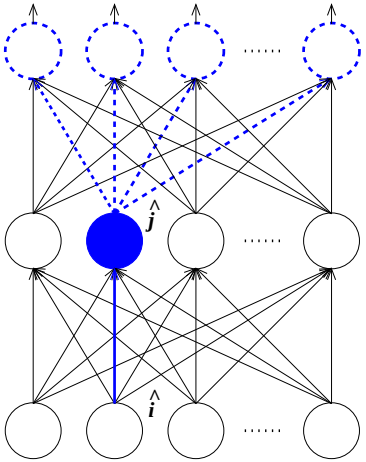


Calcolo del gradiente per i pesi di una unità nascosta

Fissiamo gli indici $\hat{j}$ e $\hat{i}$:

$$\frac{\partial E}{\partial w_{\hat{j}\hat{i}}} = \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (t_k^{(p)} - z_k^{(p)})^2$$



$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} 2(t_k^{(p)} - z_k^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (-z_k^{(p)})$$

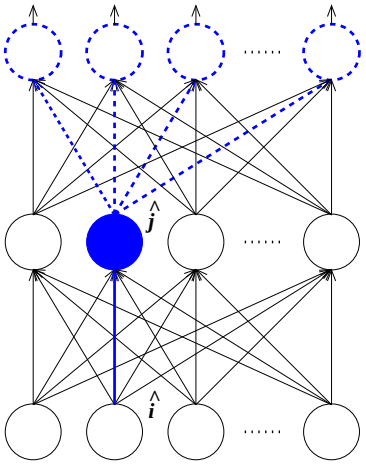
$$= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \sum_{j=1}^{N_H} w_{kj} y_j^{(p)}$$

Calcolo del gradiente per i pesi di una unità nascosta

Fissiamo gli indici $\hat{j}$ e $\hat{i}$:

$$\frac{\partial E}{\partial w_{\hat{j}\hat{i}}} = \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (t_k^{(p)} - z_k^{(p)})^2$$



$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} 2(t_k^{(p)} - z_k^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (-z_k^{(p)})$$

$$= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \sum_{j=1}^{N_H} w_{kj} y_j^{(p)}$$

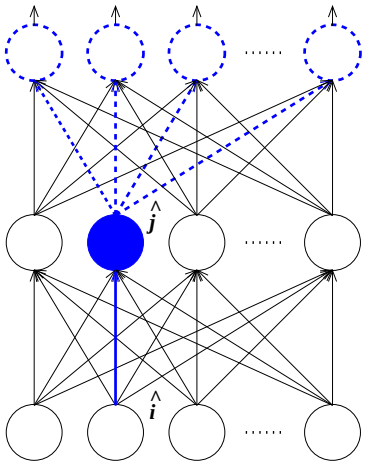
$$= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) w_{k\hat{j}} \frac{\partial}{\partial w_{\hat{j}\hat{i}}} y_{\hat{j}}^{(p)}$$

Calcolo del gradiente per i pesi di una unità nascosta

Fissiamo gli indici $\hat{j}$ e $\hat{i}$:

$$\frac{\partial E}{\partial w_{\hat{j}\hat{i}}} = \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} 2(t_k^{(p)} - z_k^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (-z_k^{(p)})$$

$$= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \sum_{j=1}^{N_H} w_{kj} y_j^{(p)}$$



$$= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) w_{k\hat{j}} \frac{\partial}{\partial w_{\hat{j}\hat{i}}} y_{\hat{j}}^{(p)}$$

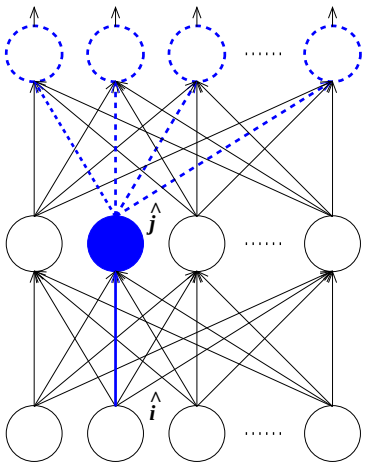
$$= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) w_{k\hat{j}} \sigma'(\vec{w}_{\hat{j}} \cdot \vec{x}^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (\vec{w}_{\hat{j}} \cdot \vec{x}^{(p)})$$

Calcolo del gradiente per i pesi di una unità nascosta

Fissiamo gli indici $\hat{j}$ e $\hat{i}$:

$$\frac{\partial E}{\partial w_{\hat{j}\hat{i}}} = \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} 2(t_k^{(p)} - z_k^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (-z_k^{(p)})$$

$$= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \sum_{j=1}^{N_H} w_{kj} y_j^{(p)}$$

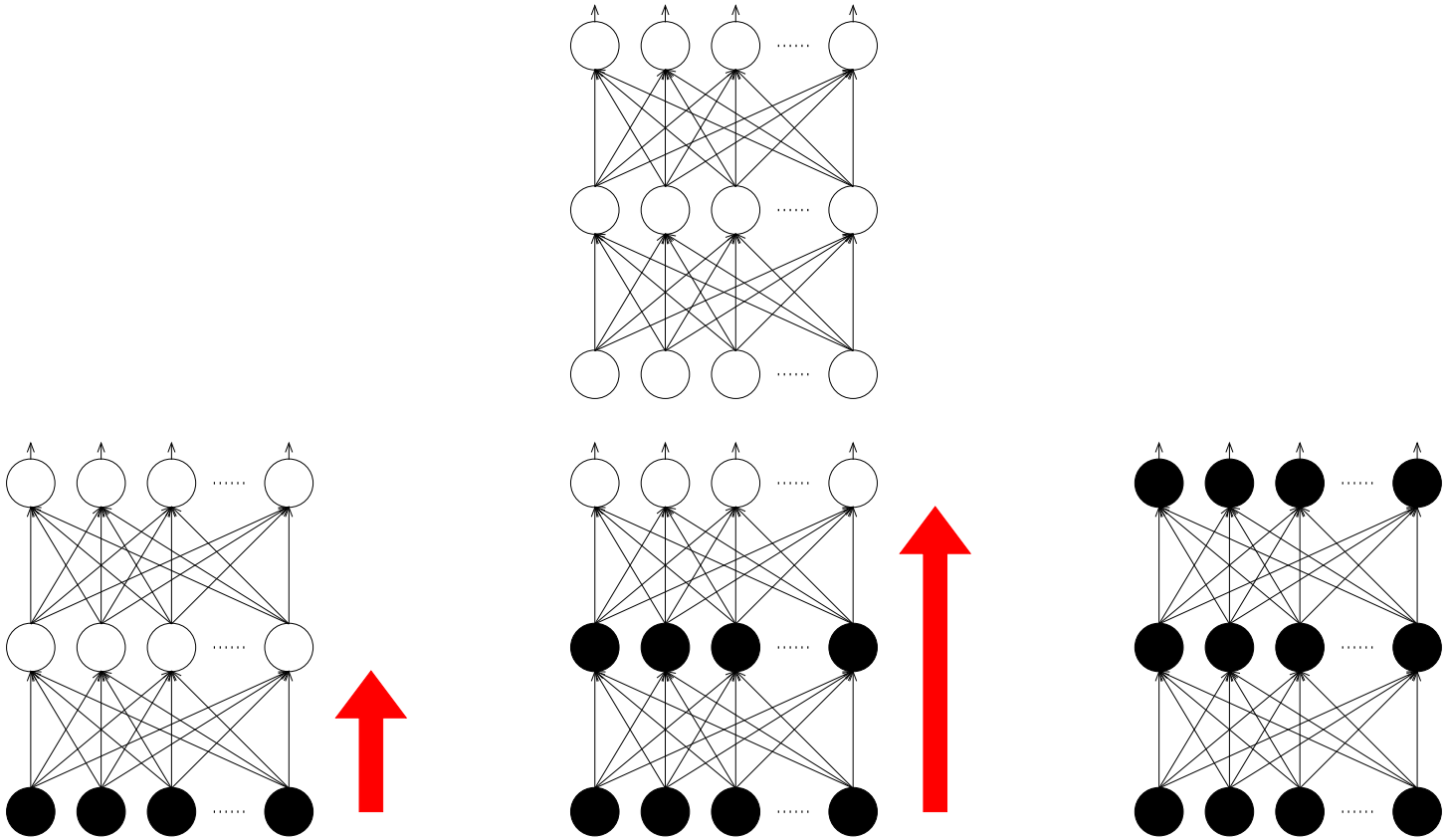


$$= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) w_{k\hat{j}} \frac{\partial}{\partial w_{\hat{j}\hat{i}}} y_{\hat{j}}^{(p)}$$

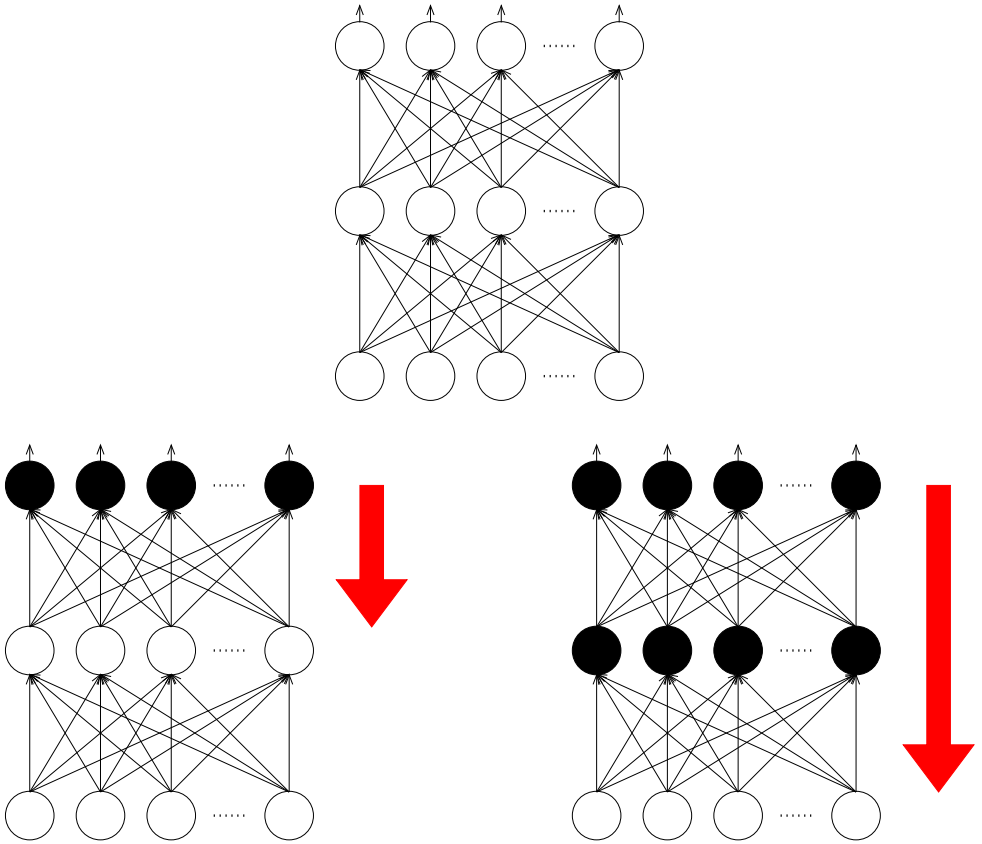
$$= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) w_{k\hat{j}} \sigma'(\vec{w}_{\hat{j}} \cdot \vec{x}^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (\vec{w}_{\hat{j}} \cdot \vec{x}^{(p)})$$

$$= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) w_{k\hat{j}} \sigma'(\vec{w}_{\hat{j}} \cdot \vec{x}^{(p)}) x_{\hat{i}}^{(p)}$$

Reti Neurali Feed-forward: Fase Forward



Reti Neurali Feed-forward: Fase Backward



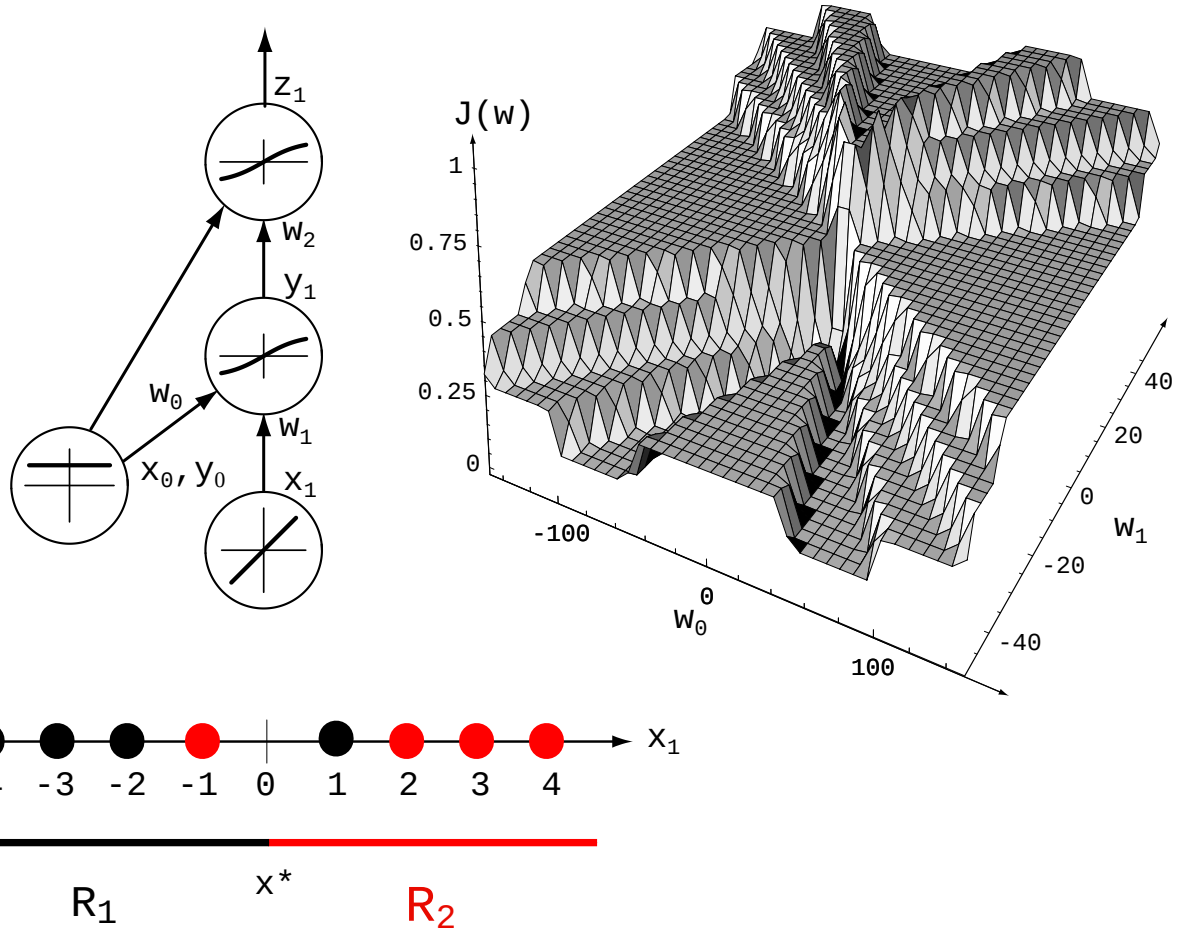
Algoritmo Back-Propagation (uno strato nascosto, stocastico)

Back-Propagation-1hl-stocastico(Tr, η, *topologia rete*)

- Inizializza tutti i pesi a valori random piccoli
- Finché la condizione di terminazione non è verifi cata, fai
  - Per ogni $(\vec{x}, \vec{t})$ in Tr, fai
    1. presenta $\vec{x}$ alla rete e calcola il corrispondente output
    2. Per ogni unità di output k
    $$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$
    3. Per ogni unità nascosta j
    $$\delta_j \leftarrow o_j(1 - o_j) \sum_{k \in outputs} w_{k,j}\delta_k$$
    4. aggiorna tutti i pesi $w_{p,q}$ della rete

$$w_{s,q} \leftarrow w_{s,q} + \eta \Delta w_{s,q} \quad \text{dove} \quad \Delta w_{s,q} = \begin{cases} \delta_s x_q & \text{se } s \in nascoste \\ \delta_s y_q & \text{se } s \in outputs \end{cases}$$

Esempio di Funzione Errore



Discesa di Gradiente Batch e Stocastica

Batch:

Fai finché condizione di terminazione non soddisfatta

1. calcola $\nabla E_{Tr}[\vec{w}]$
2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_{Tr}[\vec{w}]$

Stocastica (Incrementale):

Fai finché condizione di terminazione non soddisfatta

- Per ogni esempio di apprendimento p in Tr
  1. calcola $\nabla E_{p \in Tr}[\vec{w}]$
  2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_{p \in Tr}[\vec{w}]$

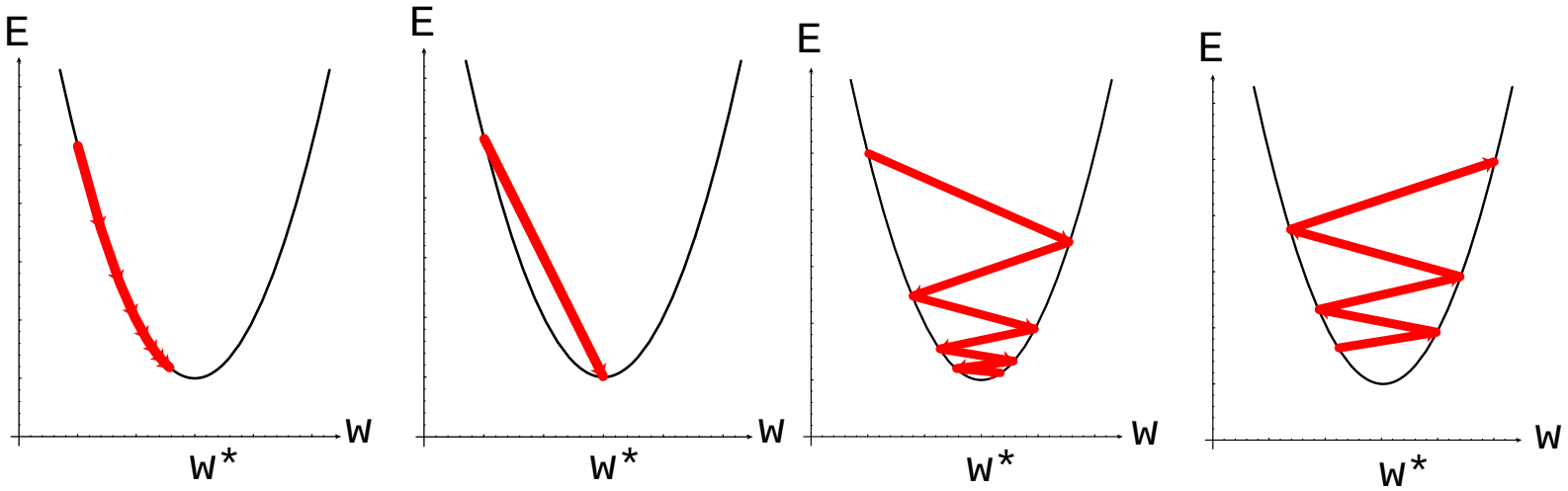
dove

$$E_{Tr}[\vec{w}] \equiv \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2 \qquad E_{p \in Tr}[\vec{w}] \equiv \frac{1}{2c} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

La discesa di gradiente *Stocastica* (gradiente istantaneo) può approssimare quella *Batch* (gradiente esatto) con precisione arbitraria se η è suffi cientemente piccolo

Alcuni Problemi ...

- Scelta della topologia della rete $\rightarrow$ determina lo Spazio delle Ipotesi;
- Scelta del passo di discesa (valore di η):



- apprendimento lento..., ma calcolo di output veloce
- MINIMI LOCALI !!

Bias Induttivo: sia nella rappresentazione che nella ricerca

Potere Computazionale Reti Neurali

Il seguente teorema stabilisce l'universalità di reti feed-forward come approssimatori di funzioni continue.

Teorema Sia $\varphi(\cdot)$ una funzione continua monotona crescente, limitata e noncostante. Si indichi con I_n l'ipercubo n-dimensionale $[0,1]^n$ e lo spazio delle funzioni continue su esso defi nite sia $C(I_n)$. Data una qualunque funzione $f \in C(I_n)$ e $\varepsilon > 0$, allora esiste un intero M e insiemi di costanti reali α_i, θ_i, e w_{ij}, dove $i = 1, \ldots, M$ e $j = 1, \ldots, n$ tale che $f(\cdot)$ possa essere approssimata da

$$F(x_1, \ldots, x_n) = \sum_{i=1}^{M} \alpha_i \varphi(\sum_{j=1}^{n} w_{ij} x_j - \theta_i) \tag{1}$$

in modo tale che

$$|F(x_1, \ldots, x_n) - f(x_1, \ldots, x_n)| < \varepsilon \tag{2}$$

per tutti i punti $[x_1, \ldots, x_n] \in I_n$.

Potere Computazionale Reti Neurali

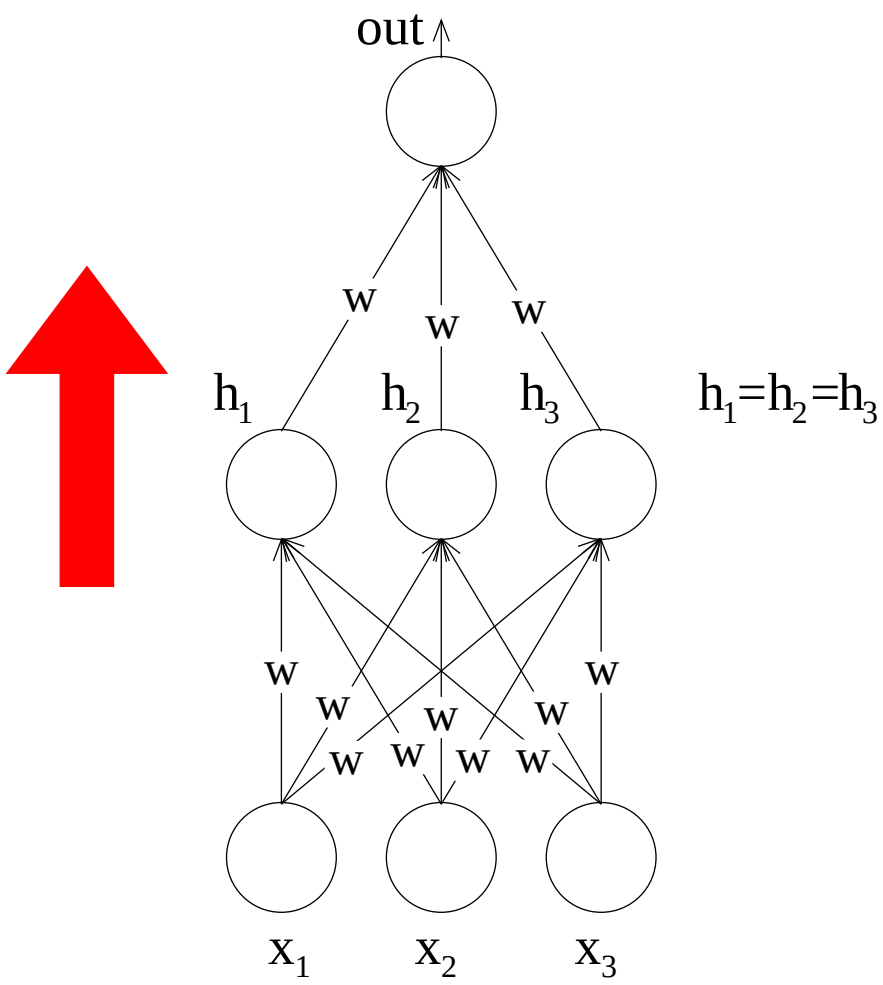
Notare che qualunque funzione sigmoidale soddisfa le condizioni imposte nel teorema su $\varphi(\cdot)$. Inoltre, l'equazione (1) rappresenta l'output di una rete multistrato descritta come segue

1. la rete ha n nodi di input ed un singolo strato di unità nascoste con M unità; gli input sono denotati da $x_1, \ldots, x_n$.
2. l'i-esima unità ha associati i pesi $w_{i1}, \ldots, w_{in}$ e soglia θ_i.
3. l'output della rete è una combinazione lineare degli output delle unità nascoste, dove i coeffi cienti della combinazione sono dati da $\alpha_1, \ldots, \alpha_M$.

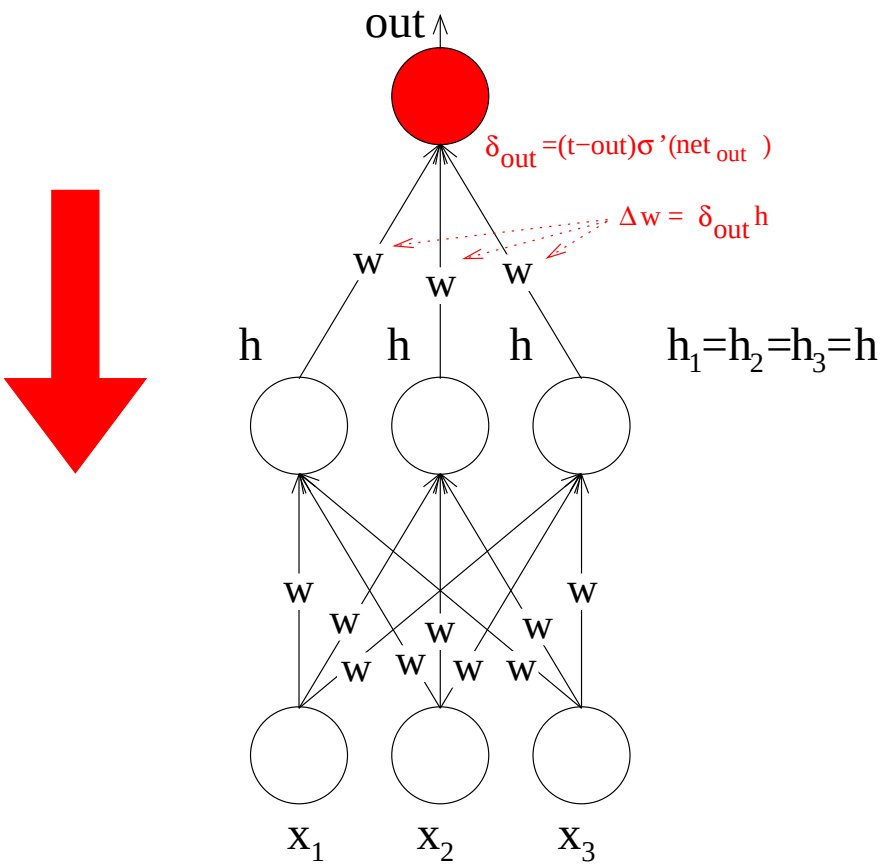
Quindi, data una tolleranza ε, una rete con un unico strato nascosto può approssimare una qualsiasi funzione in $C(I_n)$.

Si noti che il teorema afferma solo l'esistenza di una rete e non fornisce alcuna formula per il calcolo del numero M di unità nascoste necessarie per approssimare la funzione target con la tolleranza desiderata.

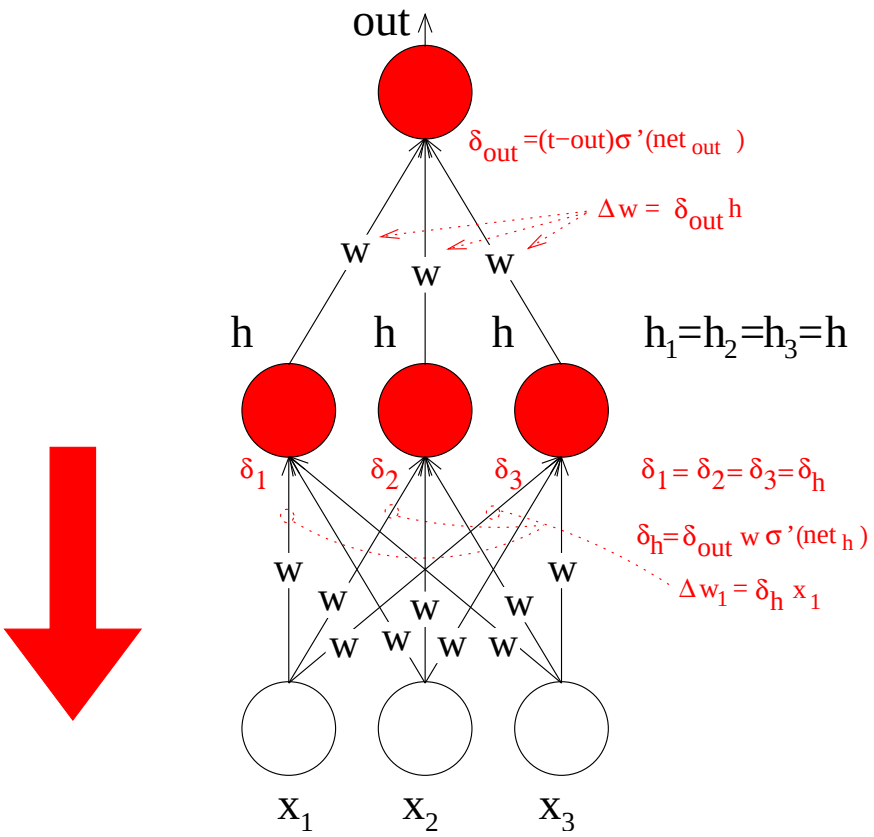
Simmetrie



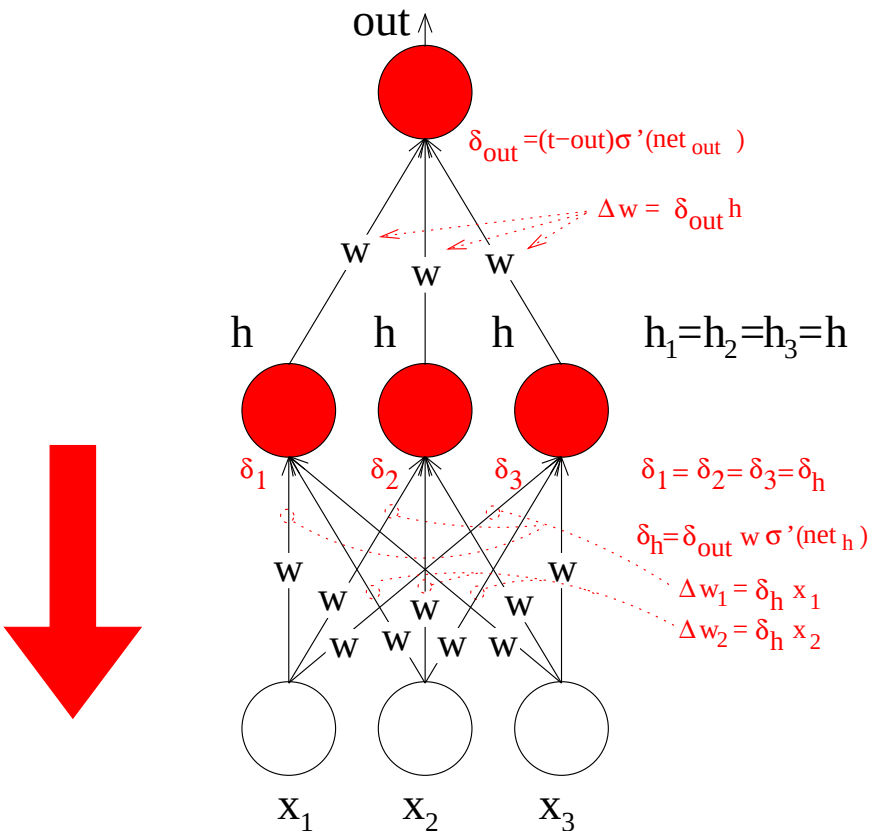
Simmetrie


out
$\delta_{out} = (t - out)\sigma'(net_{out})$
$\Delta w = \delta_{out} h$
w
h
$h_1 = h_2 = h_3 = h$
x_1 x_2 x_3

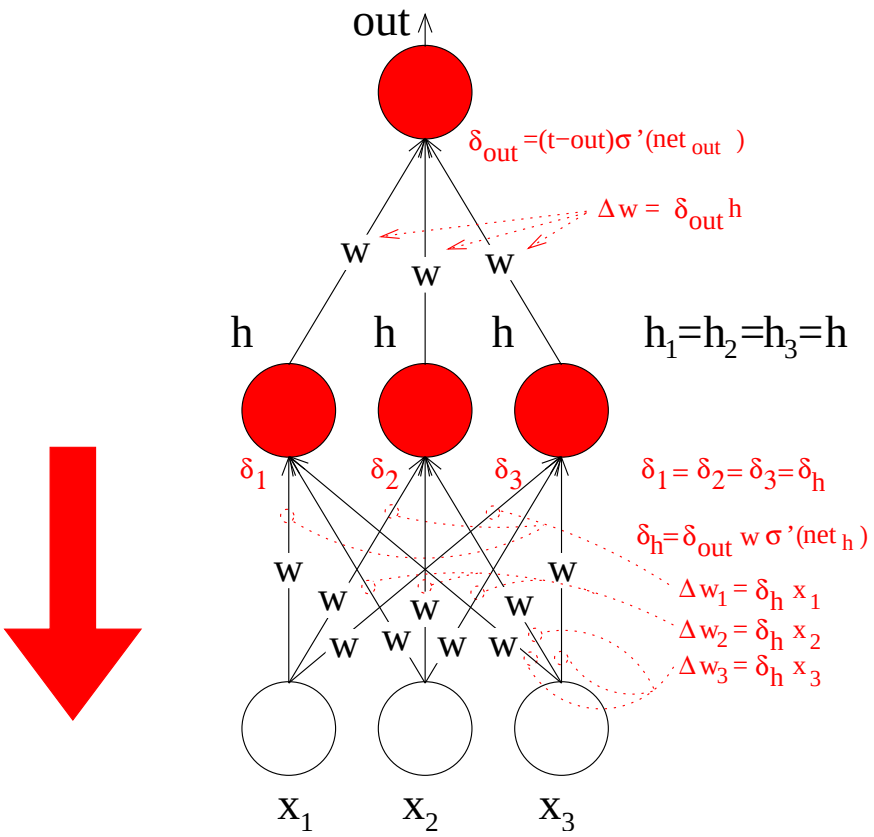
Simmetrie


out
$\delta_{out} = (t-out)\sigma'(net_{out})$
$\Delta w = \delta_{out} h$
w
h
$h_1 = h_2 = h_3 = h$
δ_1 δ_2 δ_3
$\delta_1 = \delta_2 = \delta_3 = \delta_h$
$\delta_h = \delta_{out} w \sigma'(net_h)$
$\Delta w_1 = \delta_h x_1$
x_1 x_2 x_3

Simmetrie


out
$\delta_{out}=(t-out)\sigma'(net_{out})$
$\Delta w = \delta_{out} h$
w
h
$h_1=h_2=h_3=h$
δ_1 δ_2 δ_3
$\delta_1=\delta_2=\delta_3=\delta_h$
$\delta_h=\delta_{out}\, w\, \sigma'(net_h)$
$\Delta w_1 = \delta_h x_1$
$\Delta w_2 = \delta_h x_2$
x_1 x_2 x_3

Simmetrie


out
$\delta_{out} = (t-out)\sigma'(net_{out})$
$\Delta w = \delta_{out} h$
w
h
$h_1 = h_2 = h_3 = h$
δ_1 δ_2 δ_3
$\delta_1 = \delta_2 = \delta_3 = \delta_h$
$\delta_h = \delta_{out} w \sigma'(net_h)$
$\Delta w_1 = \delta_h x_1$
$\Delta w_2 = \delta_h x_2$
$\Delta w_3 = \delta_h x_3$
x_1 x_2 x_3

Apprendimento Perceptron: prova di convergenza

Sia dato un insieme di esempi $Tr = \{(\vec{x}^{(i)}, t^{(i)})\}$, dove $t \in \{-1, +1\}$ (classificazione binaria)

Se $\exists \vec{w}^*$ t.c. (lineare separabilità)

$$\forall i,\ t^{(i)}(\vec{w}^* \cdot \vec{x}^{(i)}) \geq \delta = \min_i\ t^{(i)}(\vec{w}^* \cdot \vec{x}^{(i)}) > 0$$

allora

$$\forall i,\ \vec{w}^* \cdot (t^{(i)}\vec{x}^{(i)}) \geq \delta$$

cioè $\vec{w}^*$ è soluzione anche del problema di apprendimento definito dall'insieme di apprendimento $Tr' = \{(t^{(i)}\vec{x}^{(i)}, +1)\}$ e viceversa

Quindi ci possiamo concentrare su problemi dove **tutti gli esempi sono positivi**

Apprendimento Perceptron: prova di convergenza

Supponiamo che

- $\vec{w} = \vec{w}(0) = 0$ (inizializzazione)
- $\eta = \frac{1}{2}$ e $\forall i,\ \|\vec{x}^{(i)}\|^2 \leq K$

Dopo aver commesso q errori (tutti falsi negativi) si avrà

$$\vec{w}(q) = \sum_{j=1}^{q} \vec{x}^{(i_j)}$$

infatti all' errore j-esimo il vettore dei pesi è aggiornato sommandogli l'input $\vec{x}^{(i_j)}$ classificato erroneamente:

$$\vec{w}(j) = \vec{w}(j-1) + \vec{x}^{(i_j)}$$

Apprendimento Perceptron: prova di convergenza

Mostriamo adesso che il modulo di $\vec{w}(q)$ non può crescere indefinitivamente (succede se non si converge ad una soluzione in un numero finito di iterazioni)

Iniziamo definendo un lower bound sul modulo di $\vec{w}(q)$:

$$\vec{w}^* \cdot \vec{w}(q) = \vec{w}^* \sum_{j=1}^{q} \vec{x}^{(i_j)} \geq q\delta \quad \text{(ricordiamo che } \delta = \min_i \ \vec{w}^* \cdot \vec{x}^{(i)}\text{)}$$

e per la disuguaglianza di Cauchy-Swartz ($[\vec{x} \cdot \vec{y}]^2 \leq \|\vec{x}\|^2 \|\vec{y}\|^2$) abbiamo

$$\|\vec{w}^*\|^2 \|\vec{w}(q)\|^2 \geq [\vec{w}^* \cdot \vec{w}(q)]^2 \geq [q\delta]^2 \ \Rightarrow \ \|\vec{w}(q)\|^2 \geq \frac{[q\delta]^2}{\|\vec{w}^*\|^2}$$

Apprendimento Perceptron: prova di convergenza

Definiamo adesso un upper bound sul modulo di $\vec{w}(q)$:

$$\|\vec{w}(q)\|^2 = \|\vec{w}(q-1)+\vec{x}^{(i_q)}\|^2 = \|\vec{w}(q-1)\|^2+2\vec{w}(q-1)\cdot\vec{x}^{(i_q)}+\|\vec{x}^{(i_q)}\|^2$$

e poiché $\vec{w}(q-1)\cdot\vec{x}^{(i_q)} < 0$ (q-esimo errore)

$$\|\vec{w}(q)\|^2 \leq \|\vec{w}(q-1)\|^2 + \|\vec{x}^{(i_q)}\|^2$$

Applicando ricorsivamente questa disuguaglianza su tutti gli errori abbiamo

$$\|\vec{w}(q)\|^2 \leq \sum_{i=1}^{q} \|\vec{x}^{(i_q)}\|^2 \leq qK$$

Apprendimento Perceptron: prova di convergenza

Mettendo insieme il lower bound con l'upper bound otteniamo:

$$\frac{[q\delta]^2}{||\vec{w}^*||^2} \leq ||\vec{w}(q)||^2 \leq qK$$

Pertanto il numero massimo di errori q_{max} che si possono commettere mantenendo i due vincoli soddisfatti è ottenuto quando vale l'uguaglianza dei bound:

$$\frac{[q_{max}\delta]^2}{||\vec{w}^*||^2} = q_{max}K$$

da cui si ottiene

$$q_{max} = \frac{||\vec{w}^*||^2 K}{\delta^2}$$

Quindi, in caso di lineare separabilità, l'algoritmo di apprendimento del Perceptron commette un numero finito di errori, al più $\frac{||\vec{w}^*||^2 K}{\delta^2}$

Structural Risk Minimization

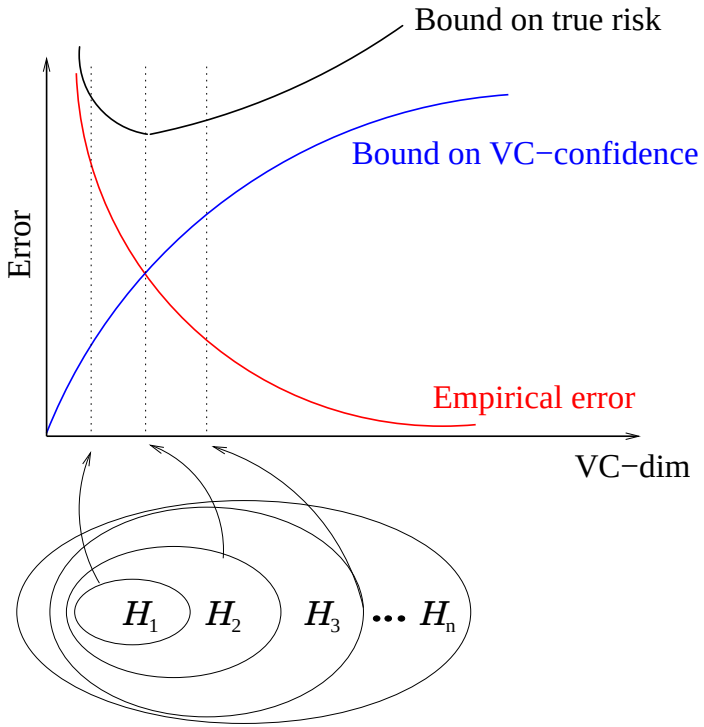
Problema: all'aumentare della VC-dimension diminuisce l'errore empirico (termine A), ma aumenta la VC confi dence (termine B)!

L'approccio Structural Risk Minimization tenta di trovare un compromesso tra i due termini:

Si considerano $\mathcal{H}_i$ tali che

- $\mathcal{H}_1 \subseteq \mathcal{H}_2 \subseteq \cdots \subseteq \mathcal{H}_n$
- $VC(\mathcal{H}_1) \leq \cdots \leq VC(\mathcal{H}_n)$
- si seleziona l'ipotesi che ha il bound sull'errore ideale pi`u basso

Esempio: Reti neurali con un numero crescente di neuroni nascosti



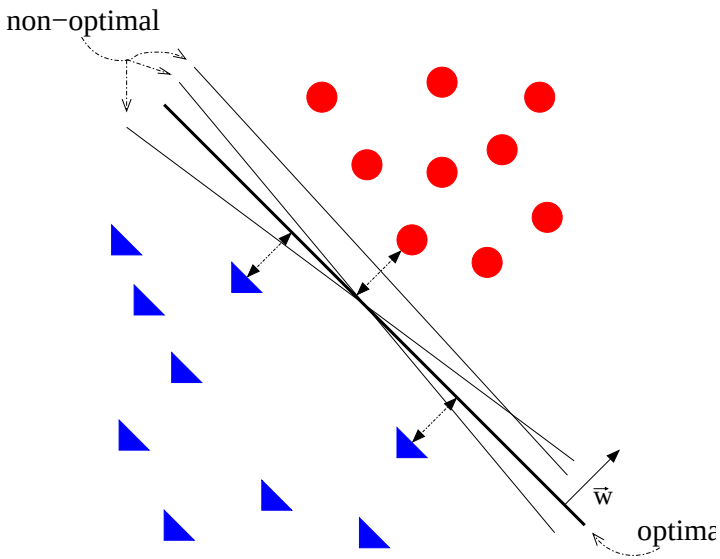
Support Vector Machines: idea base

Possiamo applicare l'approccio Structural Risk Minimization a spazi delle ipotesi costituiti da iperpiani ?

Sappiamo che un iperpiano in uno spazio a m dimensioni ha $VC = m + 1$. Come facciamo a creare una struttura di spazi delle ipotesi con VC-dimension crescente ?

Bisogna porre dei vincoli sugli iperpiani! Consideriamo iperpiani separatori con margine r

Consideriamo il caso in cui gli esempi siano linearmente separabili.
Il margine r è la "distanza" fra l'iperpiano e l'esempio più vicino.
L'iperpiano con margine maggiore è detto ottimo.



Margine

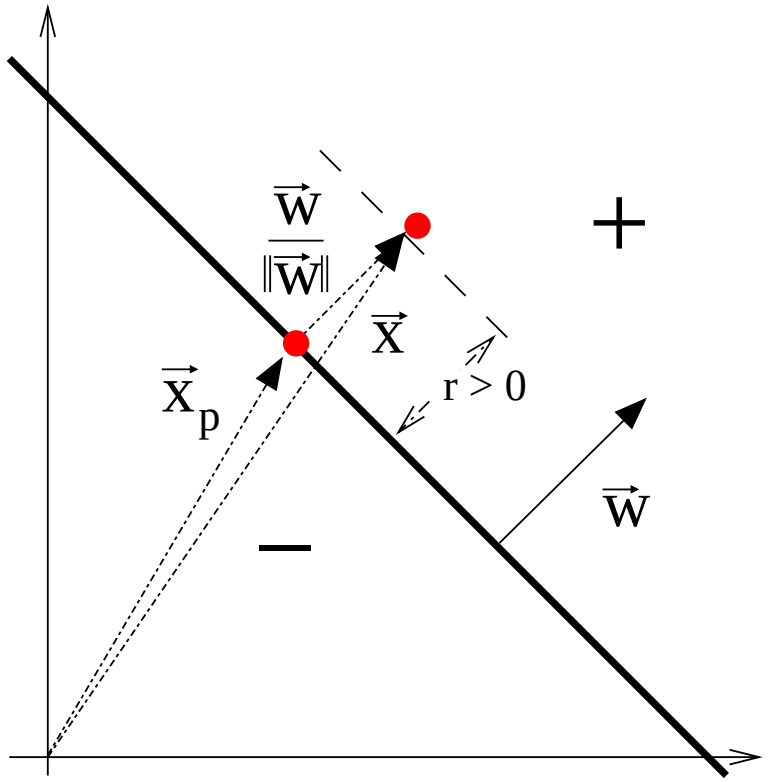
La “distanza” di un vettore da un iperpiano la possimo misurare in senso algebrico.

Dato un iperpiano determinato dalla equazione $\vec{w} \cdot \vec{x} + b = 0$, la funzione discriminante $g(\vec{x}) = \vec{w} \cdot \vec{x} + b$ restituisce la distanza algebrica di $\vec{x}$ dall’iperpiano.

Infatti, se esprimiamo $\vec{x}$ come

$$\vec{x} = \vec{x}_p + r \frac{\vec{w}}{||\vec{w}||}$$

dove $\vec{x}_p$ è la proiezione normale di $\vec{x}$ sull’iperpiano ed r è la distanza algebrica desiderata ($r > 0$ se $\vec{x}$ è sul lato positivo dell’iperpiano, altrimenti $r < 0$), allora $g(\vec{x}_p) = 0$ (poiché $\vec{x}_p$ risiede sull’iperpiano).



Quindi

$$g(\vec{x}) = \vec{w} \cdot \vec{x} + b = r||\vec{w}||$$

o meglio $r = \frac{\vec{w} \cdot \vec{x} + b}{||\vec{w}||} = \frac{g(\vec{x})}{||\vec{w}||}$

Poiché esiste una infinità di soluzioni che differiscono solo per un fattore di scala su $\vec{w}$ (si noti che l'iperpiano non cambia scalando il suo vettore normale) ci si limita per convenzione a soluzioni che soddisfano l'equazione $\hat{r}||\vec{w}|| = 1$

Si noti che per l'iperpiano ottimo, la distanza assoluta da uno degli esempi positivi più vicini è uguale a quella da uno degli esempi negativi più vicini. Il margine di separazione ρ è quindi definito come il doppio del margine: $\rho = \frac{2}{||\vec{w}||}$

Inoltre, se gli esempi sono linearmente separabili con margine $\hat{r}$ da un iperpiano, allora

$$\frac{y_i g(\vec{x}_i)}{||\vec{w}||} \geq \hat{r} \quad i = 1, \ldots, n$$

dove $y_i = 1$ per esempi positivi e $y_i = -1$ per esempi negativi. Il problema di trovare l'iperpiano ottimo si riduce quindi a quello di minimizzare $||\vec{w}||$.

Margine: Legame con SRM

Theorema *Sia R il diametro della palla più piccola che contiene tutti gli esempi di apprendimento. L'insieme di iperpiani ottimi descritti dall'equazione* $\vec{w} \cdot \vec{x} + b = 0$ *possiede VC-dimension* h *limitata superiormente da*

$$h \leq \min\{\lceil \frac{R^2}{\rho^2} \rceil, m\} + 1$$

dove $\rho = \frac{2}{||\vec{w}||}$ *ed* m *è la dimensionalità dei dati di apprendimento.*

Quindi, se consideriamo gli spazi delle ipotesi

$$\mathcal{H}_k = \{\vec{w} \cdot \vec{x} + b \mid ||\vec{w}||^2 \leq c_k\} \text{ con } c_1 < c_2 < c_3 < \ldots$$

ed i dati sono linearmente separabili, allora l'errore empirico è nullo per tutti gli iperpiani e quindi per minimizzare il bound sull'errore ideale si deve selezionare l'iperpiano con VC-dimension minima, cioè quello che minimizza $||\vec{w}||^2$ (o equivalentemente massimizza il margine di separazione).

Caso Separabile: Formulazione Quadratica

Nel caso di n esempi $\{(\vec{x}_i, y_i)\}_1^n$ linearmente separabili, è possibile trovare l'iperpiano ottimo risolvendo il seguente problema vincolato di ottimizzazione quadratica:

$$\min_{\vec{w},b} \frac{1}{2}||\vec{w}||^2$$

soggetto a: $\forall i \in \{1, \ldots, n\} : y_i(\vec{w} \cdot \vec{x}_i + b) - 1 \geq 0$

Questo problema, detto problema primale, si può risolvere più facilmente passando alla sua formulazione duale.

La teoria della ottimizzazione afferma che:

1. un problema di ottimizzazione possiede una forma duale (più semplice da risolvere) se la funzione di costo e i vincoli sono strettamente convessi;
2. se le condizioni in 1 sono soddisfatte, l'ottimo per il problema duale coincide con l'ottimo del primale.

Il nostro problema primale soddisfa le condizioni in 1.

Per passare dal primale alla sua forma duale si utilizza il teorema Kuhn-Tucker, che prescrive i seguenti due passi:

1. a partire dalla formulazione primale si costruisce un nuovo problema non vincolato utilizzando i moltiplicatori di Lagrange:

$$L(\vec{w}, b, \alpha) = \frac{1}{2}||\vec{w}||^2 - \sum_{i=1}^{n} \alpha_i (y_i(\vec{w} \cdot \vec{x}_i + b) - 1)$$

   dove le variabili $\alpha_i \geq 0$ sono i *moltiplicatori di Lagrange* (in questo caso, variabili duali). La soluzione ottima risiede nel punto di sella ottenuto minimizzando la funzione Lagrangiana $L(\vec{w}, b, \alpha)$ rispetto alle variabili primali $\vec{w}$ e b, e massimizzandola rispetto alle variabili duali α_i.

2. si utilizzano le *condizioni* di Kuhn-Tucker per esprimere le variabili primali in funzione delle variabili duali; in questo modo la funzione Lagrangiana diventa funzione esclusiva delle variabili duali e quindi deve essere massimizzata rispetto a queste variabili:

Vediamo nel dettaglio il passo 2...

Passo 2:

Il teorema Kuhn-Tucker afferma che l'ottimo si ottiene minimizzando $L(\vec{w}, b, \alpha)$ rispetto a $\vec{w}$ e b, quindi bisogna che il corrispondente gradiente della funzione sia nullo:

$$\frac{\partial L(\vec{w},b,\alpha)}{\partial \vec{w}} = 0 \quad \Leftrightarrow \quad \vec{w^*} = \sum_{i=1}^{n} y_i \alpha_i^* \vec{x}_i \text{ E1}$$
$$\frac{\partial L(\vec{w},b,\alpha)}{\partial b} = 0 \quad \Leftrightarrow \quad \sum_{i=1}^{n} y_i \alpha_i^* = 0 \text{ E2}$$

Inoltre il teorema Kuhn-Tucker afferma che all'ottimo

$$\alpha_i^* [y_i(\vec{w}^* \cdot \vec{x}_i + b^*) - 1] = 0 \;\; i = 1, \ldots, n$$

I vettori per cui $\alpha_i^* > 0$ sono detti vettori di supporto.

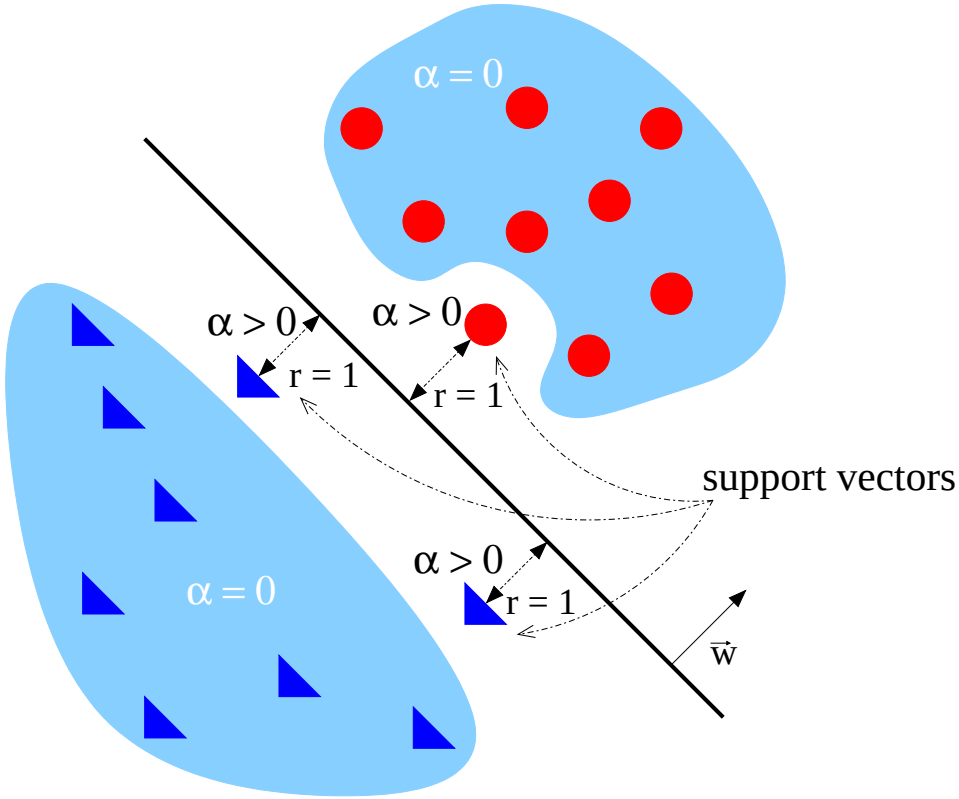
La formulazione duale si ottiene eliminando le variabili primali (equazioni E1 ed E2) dalla funzione Lagrangiana:

$$\max_\alpha \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \vec{x}_i \cdot \vec{x}_j$$
soggetto a: $\forall i \in \{1, \ldots, n\} : \alpha_i \geq 0$ e $\sum_{i=1}^{n} y_i \alpha_i = 0.$

I valori ottimi delle α_i^* determinano l'iperpiano ottimo grazie ad E1, a meno del valore di b.

Il valore di b, tuttavia, si può ottenere osservando che per un qualsiasi vettore di supporto $\vec{x}_s$ deve valere $\boxed{y_s(\vec{w}^* \cdot \vec{x}_s + b^*) = 1}$ e quindi considerando un esempio positivo ($y_s = 1$)

$$b^* = 1 - \vec{w}^* \cdot \vec{x}_s$$



Caso Non Separabile

Cosa succede se gli esempi NON sono linearmente separabili ?

In questo caso si deve ammettere che alcuni dei vincoli possano essere violati. Ciò si può fare:

- introducendo le variabili *slack* $\xi_i \geq 0,\ i = 1, \ldots, n$, una per ogni vincolo:

$$y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1 - \xi_i$$

- modificando la funzione costo in modo da penalizzare variabili slack che non sono a 0:

$$\frac{1}{2}||\vec{w}||^2 + C\sum_{i=1}^{n} \xi_i$$

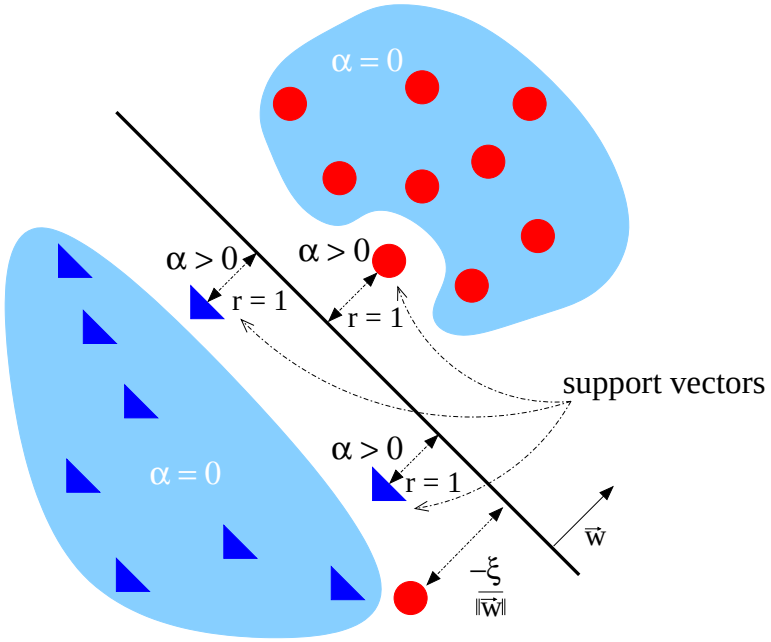
dove C (parametro di regolarizzazione) è una costante positiva che controlla il tradeoff tra la complessità dello spazio delle ipotesi e il numero di esempi non-separabili.

Il duale di questa nuova formulazione è molto simile al precedente:

$$\max_\alpha \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n y_i y_j \alpha_i \alpha_j \vec{x}_i \cdot \vec{x}_j$$

soggetto a: $\forall i \in \{1, \ldots, n\} : 0 \leq \alpha_i \leq C$ e $\sum_{i=1}^n y_i \alpha_i = 0.$

La differenza risiede nel fatto che le variabili duali sono ora limitate superiormente da C. Per la determinazione di b^* si procede in modo simile a quanto visto in precedenza (anche se con alcune differenze...).



Caso Non Separabile

La soluzione vista in precedenza per esempi non-linearmente separabili non garantisce usualmente buone prestazioni perchè un iperpiano può solo rappresentare dicotomie dello spazio delle istanze.

Per tale motivo, quando gli esempi non sono linearmente separabili si usa la seguente strategia in due passi:

1. si mappano i dati in ingresso (input space) in uno spazio a dimensione molto superiore (feature space);
2. si calcola l'iperpiano ottimo (usando la formulazione con variabili slack) all'interno del feature space.

Caso Non Separabile

Il passo 1 si giustifica tramite il teorema di Cover sulla separabilità, il quale afferma che un problema di classificazione complesso, formulato attraverso una trasformazione non-lineare dei dati in uno spazio ad alta dimensionalità, ha maggiore probabilità di essere linearmente separabile che in uno spazio a bassa dimensionalità.

Il passo 2 è ovviamente giustificato dal fatto che l'iperpiano ottimo minimizza la VC-dimension e quindi la capacità di generalizzazione è migliorata.

Il passo 1 ci prescrive di considerare una trasforamzione $\varphi(\cdot)$ non-lineare applicata ai dati originari $\{(\vec{x}_i, y_i)\}_1^n$. In particolare, assunto che $\forall i\ \vec{x}_i \in I\!R^m$, $\varphi(\cdot)$ deve mappare tali vettori (e più in generale un qualsiasi vettore a valori reali di dimensione m) in uno spazio a dimensionalità $M \gg m$ (ad esempio, $I\!R^M$).

Caso Non Separabile

Possiamo assumere che ognuna delle nuove coordinate nello spazio delle features sia generata da una funzione non-lineare $\varphi_j(\cdot)$. Quindi si considerano M funzioni $\varphi_j(\vec{x})$ con $j = 1, \ldots, M$. Un generico vettore $\vec{x}$ viene perciò mappato nel vettore M dimensionale

$$\vec{\varphi}(\vec{x}) = [\varphi_1(\vec{x}), \ldots, \varphi_M(\vec{x})]$$

Il passo 2 ci chiede di trovare un iperpiano ottimo nello spazio M dimensionale delle features. Un iperpiano in tale spazio sarà individuato dalla equazione

$$\sum_{j=1}^{M} w_j \varphi_j(\vec{x}) + b = 0$$

ovvero

$$\sum_{j=0}^{M} w_j \varphi_j(\vec{x}) = \vec{w} \cdot \vec{\varphi}(\vec{x}) = 0$$

se aggiungiamo la coordinata $\varphi_0(\vec{x}) = 1$ e $w_0 = b$.

Caso Non Separabile

Utilizzando per $\vec{w}$ la formula

$$\vec{w} = \sum_{k=1}^{n} y_k \alpha_k \vec{\varphi}(\vec{x}_k)$$

l'equazione che determina l'iperpiano diventa:

$$\sum_{k=1}^{n} y_k \alpha_k \vec{\varphi}(\vec{x}_k) \cdot \vec{\varphi}(\vec{x}) = 0$$

dove il termine $\vec{\varphi}(\vec{x}_k) \cdot \vec{\varphi}(\vec{x})$ rappresenta il prodotto scalare nel feature space fra i vettori indotti dalla k-esima istanza di apprendimento e dal vettore di input $\vec{x}$.

Funzioni Kernel

Se fosse possibile definire una funzione $K(\cdot,\cdot)$ (detta kernel) tale che

$$K(\vec{x}_k,\vec{x}) = \vec{\varphi}(\vec{x}_k)\cdot\vec{\varphi}(\vec{x}) = \sum_{j=0}^{M} \varphi_j(\vec{x}_k)\varphi_j(\vec{x}) = K(\vec{x},\vec{x}_k)$$ (funzione simmetrica)

allora, si potrebbe specificare l'iperpiano nello spazio delle features SENZA calcolare esplicitamente i vettori nello spazio delle features:

$$\sum_{k=1}^{n} y_k\alpha_k K(\vec{x}_k,\vec{x})$$

Tali funzioni kernel di fatto esistono se alcune condizioni sono soddisfatte...

Funzioni Kernel

Teorema di Mercer

Sia $K(\vec{x}, \vec{x}\prime)$ un kernel continuo e simmetrico definito nell'intervallo chiuso $\vec{a} \leq \vec{x} \leq \vec{b}$ e similarmente per $\vec{x}\prime$. Il kernel $K(\vec{x}, \vec{x}\prime)$ può essere espanso nella serie

$$K(\vec{x}, \vec{x}\prime) = \sum_{i=1}^{\infty} \lambda_i \varphi_i(\vec{x}) \varphi_i(\vec{x}\prime)$$

con $\lambda_i > 0$. Affinché tale espansione sia valida e per la sua convergenza assoluta ed uniforme, è necessario e sufficiente che la condizione

$$\int_{\vec{b}}^{\vec{a}} \int_{\vec{b}}^{\vec{a}} K(\vec{x}, \vec{x}\prime) \psi(\vec{x}) \psi(\vec{x}\prime) d\vec{x} d\vec{x}\prime \geq 0$$

sia vera per tutte le $\psi(\cdot)$ che soddisfano

$$\int_{\vec{b}}^{\vec{a}} \psi^2(\vec{x}) d\vec{x} < \infty$$

Funzioni Kernel

Quindi in sostanza una funzione kernel che soddisfa le condizioni del teorema di Mercer rappresenta un prodotto scalare in uno spazio delle features generato da una qualche trasformazione non-lineare.

Si noti che tale spazio delle features può essere infinito (vedi espansione) e che il fatto che $\forall i\ \lambda_i > 0$ implica che il kernel è definito positivo.

Esempi di funzioni kernel:

- kernel polinomiale di grado p, $(\vec{x} \cdot \vec{x}\prime + 1)^p$
- kernel radiale (radial-basis function), $\exp(-\frac{1}{2\sigma^2}\|\vec{x} - \vec{x}\prime\|^2)$

Formulazione con Kernel

Si noti, che l'introduzione di un kernel di fatto non modifica la formulazione del problema vincolato quadratico da risolvere per determinare l'iperpiano ottimo:

$$\max_\alpha \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n y_i y_j \alpha_i \alpha_j K(\vec{x}_i, \vec{x}_j)$$

soggetto a: $\forall i \in \{1, \ldots, n\} : 0 \leq \alpha_i \leq C$ e $\sum_{i=1}^n y_i \alpha_i = 0.$

dove i valori del kernel necessari sono calcolati sulle possibili coppie di vettori di allenamento ($K(\vec{x}_i, \vec{x}_j)$, con $i, j = 1, \ldots, n$) e quindi possono essere raccolti in una matrice $\boldsymbol{K} \in \mathbb{R}^n \times \mathbb{R}^n$ (simmetrica e definita positiva) denominata matrice del kernel.

Ad esempio, se si usa un kernel polinomiale di grado $p = 3$ si ha $\boldsymbol{K}_{i,j} = (\vec{x}_i \cdot \vec{x}_j + 1)^3$ e una nuova istanza $\vec{x}$ è classificata dalla funzione

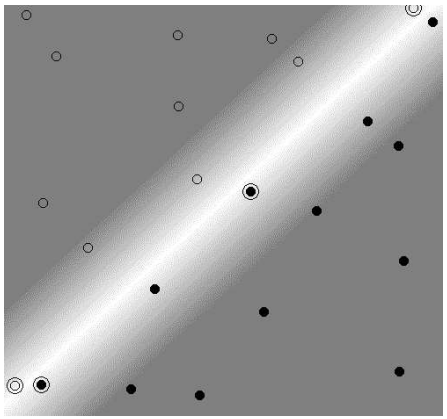
$$sign(\sum_{\vec{x}_k \in SV} y_k \alpha_k^* K(\vec{x}_k, \vec{x})) = sign(\sum_{\vec{x}_k \in SV} y_k \alpha_k^* (\vec{x}_k \cdot \vec{x} + 1)^3)$$

dove SV è l'insieme dei vettori di supporto all'ottimo e α_k^* sono i valori ottimi per i vettori di supporto (gli altri sono a 0 e quindi non contribuiscono alla sommatoria).

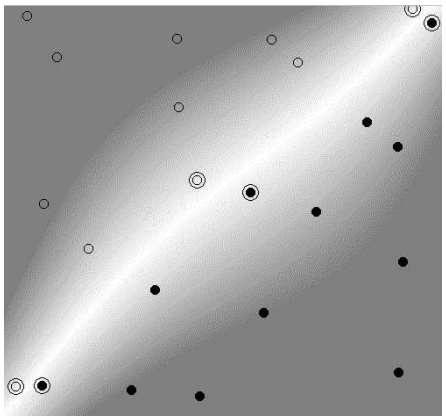
Formulazione con Kernel

Si noti come ciò permetta di effettuare la trasformazione non-lineare $\vec{\varphi}(\cdot)$ in modo IMPLICITO, in quanto quello che importa non sono i vettori nello spazio delle features, ma il prodotto scalare fra di loro, che si può calcolare direttamente tramite la funzione kernel senza passare attraverso lo spazio delle features.

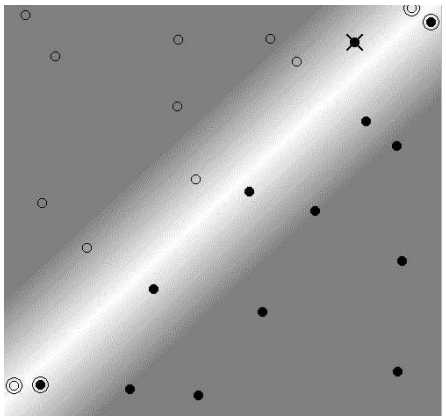
Esempi a confronto delle superfici di decisione generate NELLO SPAZIO DELLE ISTANZE senza e con kernel (polinomiale di grado 3) sia nel caso separabile che non-separabile:



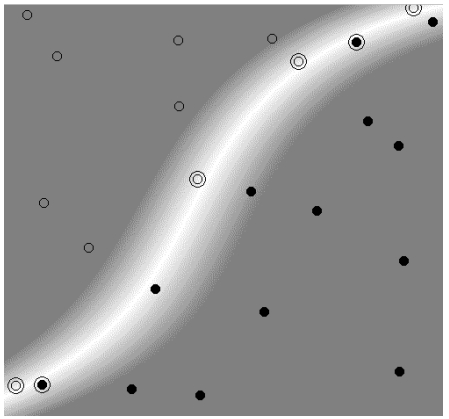
separable



degree 3 poly



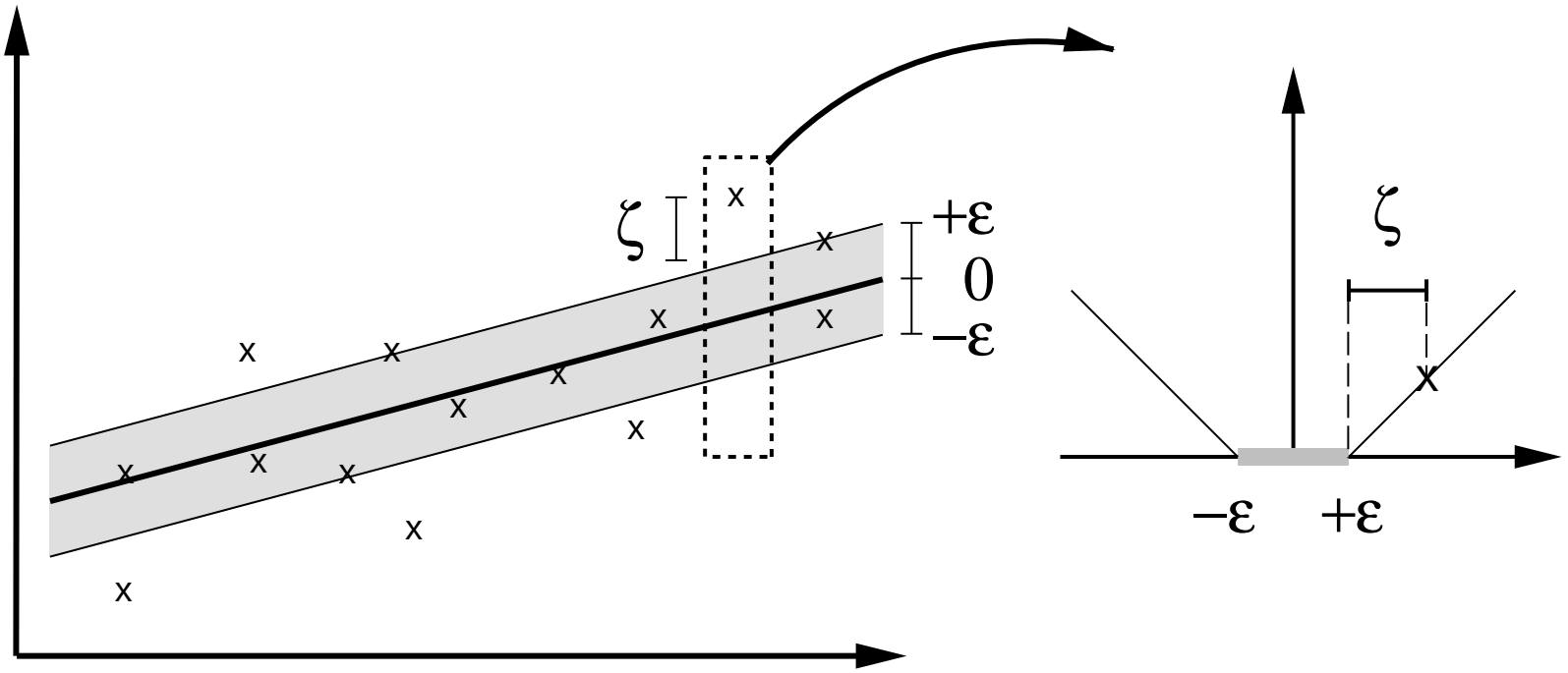
not separable



degree 3 poly

Regressione: Idea Base

Quando si considera il problema di approssimazione di funzioni a valori reali (regressione) si utilizza l'ϵ-tubo: output che differiscono dal valore di target per più di ϵ in valore assoluto vengono penalizzati linearmente, altrimenti non vengono considerati errori.



Regressione: Forma Primale

Questa idea da' origine alla seguente formulazione primale

$$\min_{\vec{w},b,\xi,\xi^*} \frac{1}{2}||\vec{w}||^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*)$$

soggetto a:

$$\forall i \in \{1,\ldots,n\}$$

$$y_i - \vec{w}\cdot\vec{x}_i - b \leq \epsilon + \xi_i$$

$$\vec{w}\cdot\vec{x}_i + b - y_i \leq \epsilon + \xi_i^*$$

$$\xi_i, \xi_i^* \geq 0$$

la cui forma duale ...

Regressione: Forma Duale

... è la seguente

$$\max_{\alpha,\alpha^*} -\epsilon \sum_{i=1}^{n}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{n} y_i(\alpha_i - \alpha_i^*) +$$
$$-\frac{1}{2}\sum_{i,j=1}^{n}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)K(\vec{x}_i, \cdot\vec{x}_j)$$

soggetto a:

$$\sum_{i=1}^{n}(\alpha_i - \alpha_i^*) = 0$$

$$\alpha_i, \alpha_i^* \in [0, C]$$

Apprendimento con Rinforzo

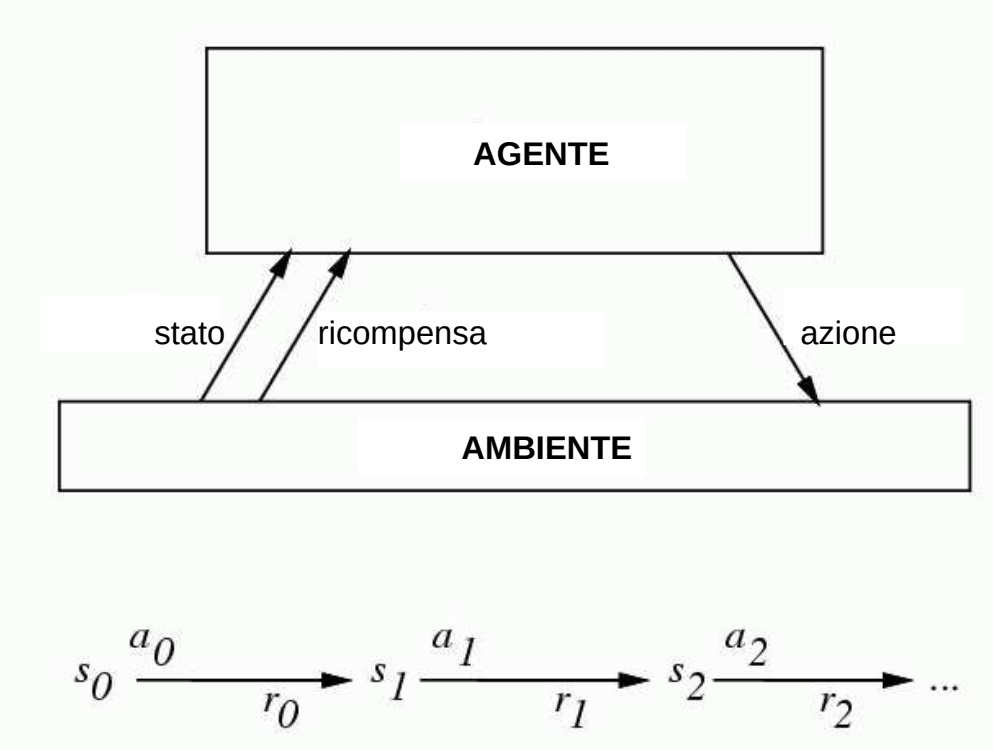
- Sono dati:
  - agente (intelligente ?), che può
    - trovarsi in uno stato s, ed
    - eseguire una azione a (all'interno delle azioni possibili nello stato corrente)
  - ed opera in un ambiente e, che applicando una azione a nello stato s restituisce
    - lo stato successivo, e
    - una ricompensa r, che può essere positiva (+), negativa (-), o neutra (0).
- Scopo dell'agente è quello di massimizzare una funzione delle ricompense
  (es. ricompensa scontata: $\sum_{t=0}^{\infty} \gamma^t r_{t+1}$ dove $0 \leq \gamma < 1$)

Esempio di applicazione: sistema che imparare a giocare a dama

Esempio di applicazione: modulo di navigazione per un robot in un ambiente sconosciuto

Esempio di applicazione: navigare sul Web alla ricerca di informazione focalizzata

Agente/Ambiente



Goal: apprendere le azioni che massimizzano

$r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$, dove $0 \leq \gamma < 1$

Processi di Decisione di Markov

Si assume

- insieme finito di stati S
- insieme di azioni A
- ad ogni istante di tempo t l'agente osserva lo stato $s_t \in S$ e sceglie l'azione $a_t \in A$
- poi riceve la ricompensa immediata r_t
- e cambia lo stato in s_{t+1}
- assunzione di Markov: $s_{t+1} = \delta(s_t, a_t)$ e $r_t = r(s_t, a_t)$
  - cioe', r_t e s_{t+1} dipendono SOLO dallo stato ed azione *correnti*
  - le funzioni δ e r possono essere DETERMINISTICHE o NONDETERMINISTICHE
  - le funzioni δ e r possono essere CONOSCIUTE o IGNOTE

Compito di Apprendimento per l'Agente

Eseguire azioni nell'Ambiente, osservare i risultati e

- apprendere la politica (per le azioni) $\pi : S \rightarrow A$ che massimizza (nel caso più generale)

$$E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots]$$

  per ogni stato iniziale in S

- $0 \leq \gamma < 1$ è il *fattore di sconto* per ricompense future

Novità:

- la funzione target è $\pi : S \rightarrow A$
- ma non si hanno a disposizione esempi nella forma (s, a)
- gli esempi di apprendimento sono nella forma $((s, a), r)$

La Funzione di Valutazione

Consideriamo il CASO DETERMINISTICO:

- per ogni possibile politica π che l'agente può adottare, possiamo definire una funzione di valutazione sugli stati

$$V^{\pi}(s) \equiv r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots$$
$$\equiv \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

  dove $r_t, r_{t+1}, \ldots$ sono generati seguendo la politica π a partire dallo stato s

- riformulato, il compito consiste nell'apprendere la politica ottima π^*

$$\pi^* \equiv arg\max_{\pi} V^{\pi}(s), (\forall s)$$

Cosa Apprendere ?

Possiamo tentare di far apprendere all'agente la funzione di valutazione V^{π^*} (riferita nel seguito semplicemente con V^*)

L'agente potrebbe effettuare una ricerca in avanti per scegliere la migliore azione a partire da ogni stato s perché

$$\pi^*(s) = arg\max_a [r(s,a) + \gamma V^*(\delta(s,a))]$$

Problema:

- Funziona solo se l'agente conosce $\delta : S \times A \to S$, e $r : S \times A \to \Re$
- ... ma se l'agente non ha questa conoscenza (come succede in molte applicazioni del mondo reale) non si può procedere in questo modo...

La Funzione $Q()$

Soluzione:

- definire una nuova funzione molto simile a V^*

$$Q(s,a) \equiv r(s,a) + \gamma V^*(\delta(s,a))$$

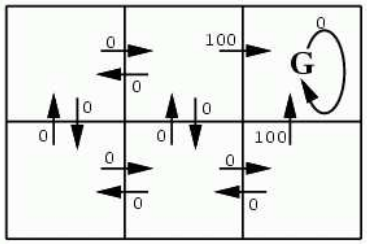
- se l'agente apprende Q, esso può scegliere l'azione ottima anche senza conoscere δ!

$$\pi^*(s) = arg\max_a [r(s,a) + \gamma V^*(\delta(s,a))]$$

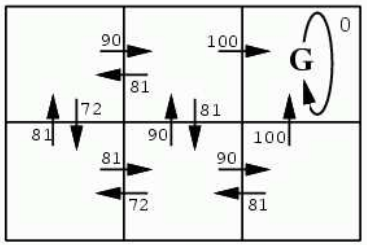
$$\pi^*(s) = arg\max_a Q(s,a)$$

Quindi, Q è la funzione di valutazione che l'agente deve apprendere

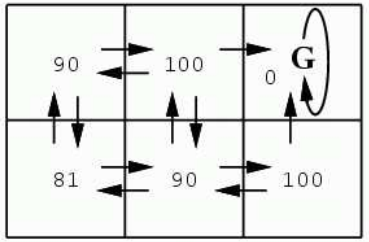
Esempio



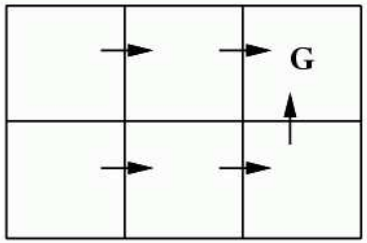
$r(s,a)$



$Q(s,a)$



$V^*(s)$



una politica ottima

Come Apprendere $Q()$?

Notare che Q e V^* sono intimamente correlate:

$$V^*(s) = \max_{a'} Q(s, a')$$

Ciò permette di scrivere Q RICORSIVAMENTE come

$$\begin{aligned} Q(s_t, a_t) &= r(s_t, a_t) + \gamma V^*(\delta(s_t, a_t)) \\ &= r(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a') \end{aligned}$$

Bene! Poniamo $\hat{Q}$ essere la funzione corrente appresa dall'agente che approssima Q.

Consideriamo la seguente regola di apprendimento:

$$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$$

dove s' è lo stato risultante dalla applicazione della azione a allo stato s

Algoritmo $Q - Learning$

1. per ogni s, a inizializza la entry della tabella $\hat{Q}(s, a) \leftarrow 0$
2. osserva lo stato corrente s
3. esegui per sempre
   (a) seleziona una azione a ed eseguirla
   (b) ricevi la ricompensa immediata r
   (c) osserva il nuovo stato s'
   (d) aggiorna la entry $\hat{Q}(s, a)$ come segue:

$$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$$

   (e) $s \leftarrow s'$

Algoritmo $Q - Learning$

1. per ogni s, a inizializza la entry della tabella $\hat{Q}(s, a) \leftarrow 0$
2. osserva lo stato corrente s
3. esegui per sempre
   (a) seleziona una azione a ed eseguila — come selezionarla ?
   (b) ricevi la ricompensa immediata r
   (c) osserva il nuovo stato s'
   (d) aggiorna la entry $\hat{Q}(s, a)$ come segue:

$$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$$

   (e) $s \leftarrow s'$

Algoritmo $Q - Learning$

1. per ogni s, a inizializza la entry della tabella $\hat{Q}(s, a) \leftarrow 0$
2. osserva lo stato corrente s
3. esegui per sempre

   (a) seleziona una azione a ... strategia → random (esplorazione); $arg\max_a \hat{Q}(s, a)$ (sfruttamento)

   (b) ricevi la ricompensa immediata r

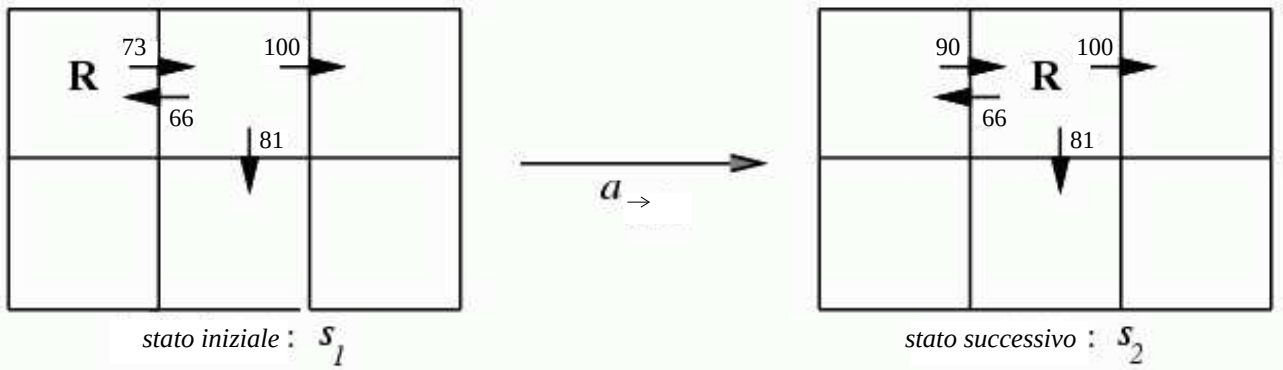
   (c) osserva il nuovo stato s'

   (d) aggiorna la entry $\hat{Q}(s, a)$ come segue:

$$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$$

   (e) $s \leftarrow s'$

Esempio di Applicazione



$$\hat{Q}(s_1, a_{\rightarrow}) \leftarrow r + \gamma \max_{a'} \hat{Q}(s_2, a')$$

$$\leftarrow 0 + 0.9 \max\{66, 81, 100\}$$

$$\leftarrow 90$$

notare che se le ricompense sono non-negative, allora

$$(\forall s, a, n)\ \hat{Q}_{n+1}(s, a) \geq \hat{Q}_n(s, a)$$

e

$$(\forall s, a, n)\ 0 \leq \hat{Q}_n(s, a) \leq Q(s, a)$$

Convergenza

$\hat{Q}$ converge a Q

Consideriamo il caso deterministico dove ogni (s, a) è visitato un numero infinito di volte

Prova: Definiamo un intervallo pieno un intervallo durante il quale ogni (s, a) è visitato. Durante ogni intervallo pieno l'errore più grande nella tabella $\hat{Q}$ è ridotto di un fattore γ

Infatti, sia $\hat{Q}_n$ la tabella dopo n aggiornamenti, e Δ_n l'errore massimo in $\hat{Q}_n$, cioè

$$\Delta_n = \max_{s,a} |\hat{Q}_n(s, a) - Q(s, a)|$$

Per ogni entry della tabella $\hat{Q}_n(s, a)$ aggiornata alla iterazione $n + 1$, l'errore nella stima rivista $\hat{Q}_{n+1}(s, a)$ è

$$
\begin{aligned}
|\hat{Q}_{n+1}(s,a) - Q(s,a)| &= |(r + \gamma \max_{a'} \hat{Q}_n(s',a')) \\
&\quad -(r + \gamma \max_{a'} Q(s',a'))| \\
&= \gamma |\max_{a'} \hat{Q}_n(s',a') - \max_{a'} Q(s',a')| \\
&\leq \gamma \max_{a'} |\hat{Q}_n(s',a') - Q(s',a')| \\
&\leq \gamma \max_{s'',a'} |\hat{Q}_n(s'',a') - Q(s'',a')| \\
|\hat{Q}_{n+1}(s,a) - Q(s,a)| &\leq \gamma \Delta_n
\end{aligned}
$$

Notare che abbiamo usato il risultato generale

$$
|\max_a f_1(a) - \max_a f_2(a)| \leq \max_a |f_1(a) - f_2(a)|
$$

Caso NONDETERMINISTICO

Si ridefiniscono V, Q considerando i valori aspettati

$$\begin{aligned} V^{\pi}(s) &\equiv E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots] \\ &\equiv E[\sum_{i=0}^{\infty} \gamma^i r_{t+i}] \end{aligned}$$

$$Q(s,a) \equiv E[r(s,a) + \gamma V^*(\delta(s,a))]$$

Per l'apprendimento, sostituire la regola di aggiornamento con

$$\hat{Q}_n(s,a) \leftarrow (1-\alpha_n)\hat{Q}_{n-1}(s,a) + \alpha_n[r + \gamma \max_{a'} \hat{Q}_{n-1}(s',a')] \text{ dove } \alpha_n = \frac{1}{1 + visite_n(s,a)}$$

Sotto determinate condizioni si dimostra la convergenza