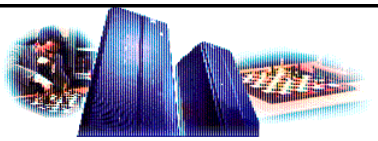
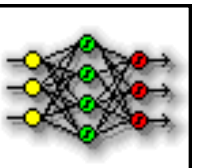
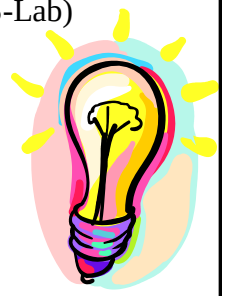




Le reti neurali

Alberto Borghese

Università degli Studi di Milano
Laboratory of Applied Intelligent Systems (AIS-Lab)
Dipartimento di Informatica
alberto.borghese@unimi.it



Sommario

Brains cause minds (J. Searle)

Le reti neurali

Se il neurone biologico consente l'intelligenza, perché non dovrebbe consentire l'intelligenza artificiale un neurone sintetico?

".. a neural network is a system composed of ***many simple processing elements*** operating in ***parallel*** whose function is determined by ***network structure***, ***connection strengths***, and the ***processing performed at computing elements*** or nodes. … Neural network architectures are inspired by the architecture of biological nervous systems, which use many simple processing elements operating in parallel to obtain high computation rates". (DARPA, 1988)….

Now, this is called learning with Kernels

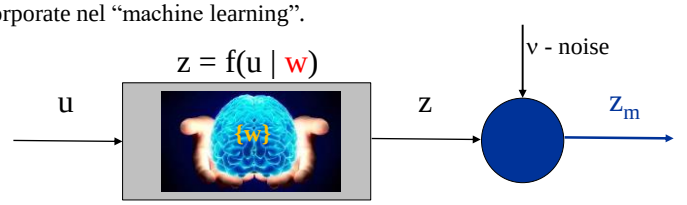
A cosa servono?

Le reti neurali offrono i seguenti specifici vantaggi nell'elaborazione dell'informazione:

- Apprendimento basato su esempi (non è richiesta l'elaborazione di un modello aderente alla realtà)
- Autoorganizzazione dell'informazione nella rete (explainability is an issue… but it is also an issue for natural neural networks!)
- Robustezza ai guasti (codifica ridondante dell'informazione)
- Funzionamento in tempo reale (realizzazione HW)
- Basso consumo (0.5nW ÷ 4nW per neurone, 20W per il SN) -> calcolatori chimici.
- Modelli di reti neurali per il calcolo.
- Interfacce artificiali con le reti neurali naturali

Cosa sono le reti neurali artificiali?

- Le reti neurali sono modelli lineari o non lineari per l'**approssimazione** della soluzione di problemi dei quali non esiste un modello con parametri semantici (o se esiste è troppo oneroso computazionalmente). I parametri dei modelli risultanti (semiparametrici) vengono calcolati mediante l'utilizzo di esempi (dati di ingresso e uscita desiderata). Connessioni con il dominio della statistica.
- Vengono utilizzate soprattutto per la classificazione e la regressione.
- Sono un capitolo importante negli argomenti di intelligenza artificiale.
- Da un altro punto di vista possono essere utilizzate per lo studio delle reti neurali naturali, ovvero dei processi cognitivi.
- Sono state incorporate nel "machine learning".



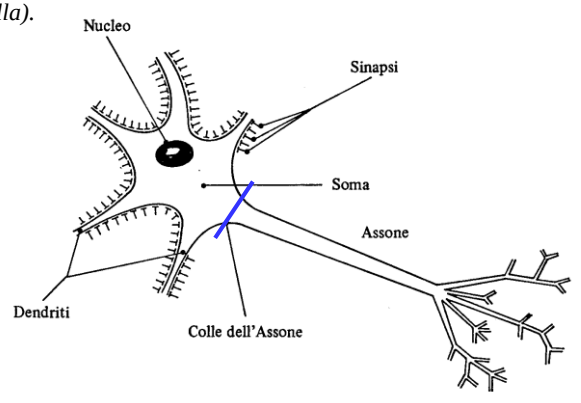
Direzione di sviluppo nell'AI

Deep neural networks for strong position on AI
- Capsule networks
- Transformers

ChatGPT for weak position on AI

Il neurone artificiale

•*Potenziale di azione (tutto o nulla).*
•*Integrazione nel soma.*
•*Soglia di attivazione.*



Neurone come elemento di calcolo universale: in grado di calcolare qualsiasi funzione logica (cioè implementabile in un computer).

Il modello di McCulloch-Pitts

•La variazione delle forma d'onda del potenziale di membrana lungo il dendrita non viene considerata.

• Gli input non sono sincroni.

• Le interazioni tra input non sono lineari.

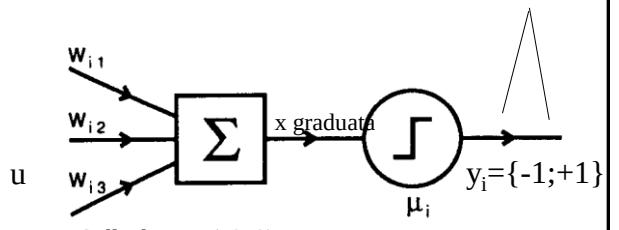
• I pesi sono supposti costanti.

$$y_i(t+1) = \Theta\left(\sum (w_{ij}u_j(t)) - \mu_i\right)$$

$$\Theta(x) = \begin{cases} 1 & se\ x \geq 0 \\ -1 & altrimenti \end{cases}$$

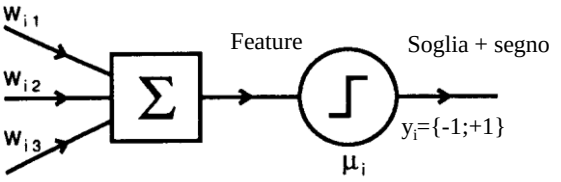
Sono state pensate per calcolare funzioni logiche (V o F).

Usate oggi per i **classificatori binari.**



McCulloch-Pitts (1943)

Algoritmi di boosting



Committee di classificatori binari

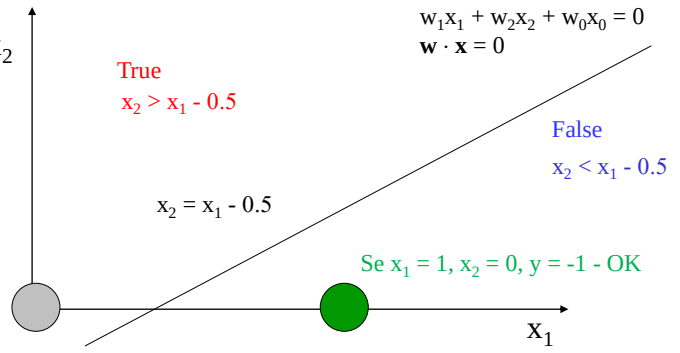
Votazione a maggioranza

Rappresentazione della retta

*La retta taglia in due il piano (**classificatore lineare**)*

$x_0 = -1$
$w_0 = \mu$

Se:
$x_1 = 0$
$x_2 = 0$
$y = +1$



$w_1x_1 + w_2x_2 - \mu = 0$

$$y_i(t+1) = \Theta\left(\sum (w_{ij}u_j(t)) - \mu_i\right)$$

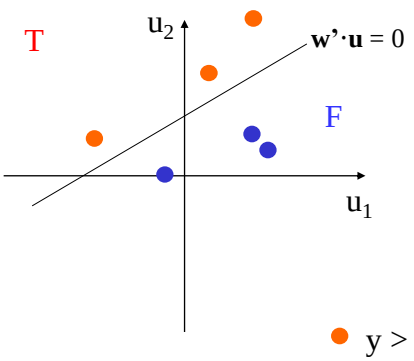
$w_1x_1 + w_2x_2 + w_0x_0 = 0$

$$y_i(t+1) = \Theta\left(\sum w_{ij}u_j(t)\right)$$

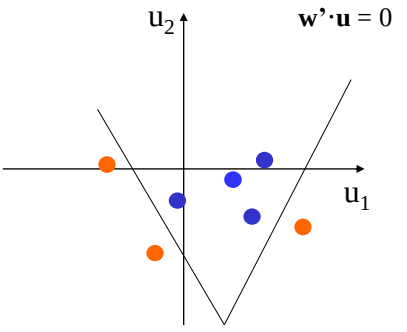
Dati linearmente separabili

$$w_1u_1 + w_2u_2 - \mu = 0$$

Linearmente separabile



Non linearmente separabile



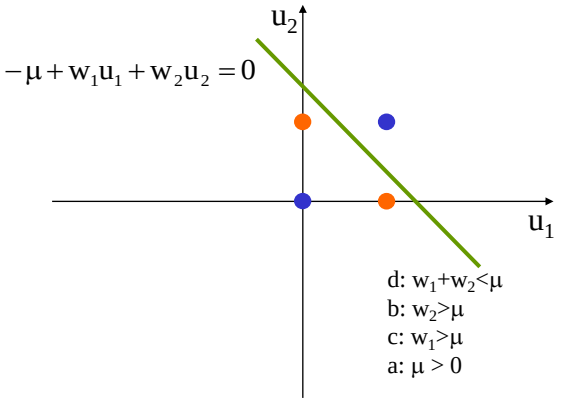
$y = f(u_1,u_2)$

$y > 0$
$y < 0$

Rilevante per i **classificatori** (boosting, SVM)
Classificare vuol dire trovare la linea di separazione tra i dati (spezzata)

La "morte" del neurone di McCulloch-Pitts (Minski, 1969): XOR

$w_1u_1 + w_2u_2 - \mu = 0$ $\quad \mathbf{w'}\cdot\mathbf{u} = 0 \quad u_0 = -1 \quad w_0 = -\mu$



d: $w_1+w_2<\mu$
b: $w_2>\mu$
c: $w_1>\mu$
a: $\mu > 0$

u_1	u_2	y	
-1	-1	-1	a
-1	1	1	b
1	-1	1	c
1	1	-1	d

Il sistema di 4 equazioni non è risolubile.

- $y(u_1, u_2, 1)=1$
- $y(u_1, u_2, 1)=-1$

$w_1,w_2> \mu$ e $w_1+w_2< \mu$

Impossibile!!

Si possono imparare solamente funzioni linearmente separabili

Sommario

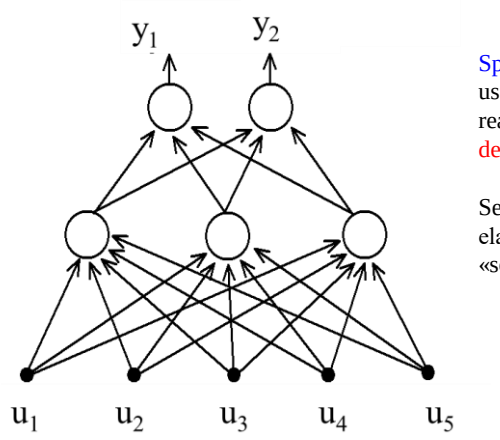
Dal neurone artificiale alle reti neurali

L'apprendimento in reti di perceptroni

Esempio con unità lineari

Spiking neurons

y^D_1 y^D_2

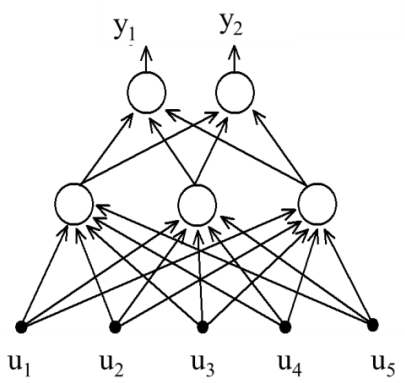


Spiking neurons. Sono neuroni la cui uscita è il singolo spike. Modellazione realistica (e.g. McCullochPitts). Spike del neurone.

Segnale binario: spike / non spike + elaborazione continua all'interno del «soma».

Neuroni artificiali «moderni»

y^D_1 y^D_2



Connessionismo classico. Uscita compresa tra min – Max. Tra 0 e 1 (o tra -1 e 1): $y_i = [0\ 1]$.

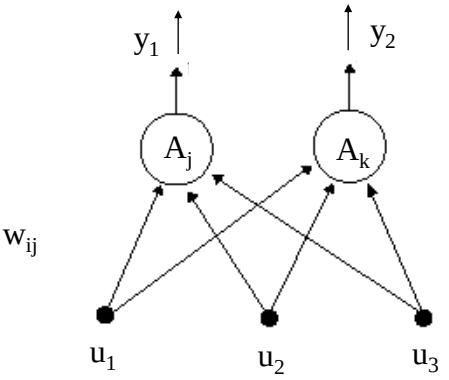
Interpretazione: frequenza di scarica. Viene trascurata la dinamica di attivazione dei neuroni.

L'uscita può essere binarizzata definendo una soglia.

L'input può essere compreso tra –oo e +oo o limitato a seconda delle unità utilizzate in input.

Lo spirito dell'apprendimento supervisionato

La rete opera una trasformazione dallo spazio di input allo spazio di output.

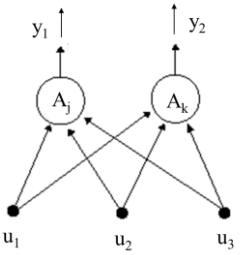


Apprendimento è la modifica dei parametri $\{w_{ij}\}$ e $\{\mu_j\}$ in modo tale che la rete neurale approssimi la trasformazione tra i pattern di input e di output.

$$y_i = g(\sum_j (w_{ij} u_j) - \mu_i)$$

Funzione costo per unità di attivazione continue

Possiamo derivare una modalità di apprendimento.
Consideriamo un perceptrone ad un livello.



$$y = g\left(\sum_{j=1} w_{ij} u_j - \mu_i\right) = g\left(\sum_{j=0} \left(w_{ij} u_j\right)\right)$$

Incorporo μ nei parametri {w}: $u_0 = -1$ $w_0 = -\mu$

Si tratta di un problema di minimizzazione di una cifra di merito, E(.), sullo spazio dei parametri W:

$$E(\mathbf{w} \mid Y^D, U) = \underbrace{\| Y^D - g(W^{nuovo}\, U)\|}_{\text{Errore}} \leq \| Y^D - g(W^{vecchio}\, U)\|$$

Devo trovare {w} : E(w) è minimo.

$$E(\mathbf{w}) = \frac{1}{2} \sum_p \left[\sum_j \left(y_{jp}^D - y_{jp}\right)^2 \right] = \frac{1}{2} \sum_p \left[\sum_j \left(y_{jp}^D - g\left(\sum_i w_{ij} u_{ip} \right) \right) \right]^2$$

su tutti i pattern p

Apprendimento supervisionato

$\min_{\{w\}} E(.)$ $\quad E(\mathbf{w} \mid Y^D, U) = \| Y^D - g(W^{nuovo}\, U)\| \leq \| Y^D - g(W^{vecchio}\, U)\|$

Y^D è l'uscita desiderata nota.

Si tratta di un problema di minimizzazione di una cifra di merito (E(.)) nello spazio dei parametri W.

Soluzione iterativa (gradiente):
Obbiettivo: se esiste una soluzione, trovare ΔW in modo iterativo tale che l'insieme dei pesi W^{nuovo} ottenuto come:

$$W^{nuovo} = W^{vecchio} + \Delta W$$

dia luogo a un errore sulle uscite di norma minore che con $W^{vecchio}$ (si parte da un W_0 iniziale, arbitrario).

Minimizzazione di funzioni di più variabili

min(E{**w**}| …..) funzione costo o errore

Gradiente: $\nabla E(w) = \frac{\delta E(\{w\}|\ldots)}{dw_1} + \frac{\delta E(\{w\}|\ldots)}{dw_2} + \frac{\delta E(\{w\}|\ldots)}{dw_3} + \frac{\delta E(\{w\}|\ldots)}{dw_4} + \ldots$

Modifico il valore dei pesi di una quantità proporzionale alla pendenza della funzione costo rispetto a quel parametro.
Estensione della tecnica del gradiente a più variabili.

$$\mathbf{\Delta w = -\eta\, \nabla E(w)} \quad \Leftrightarrow \quad \Delta w_{ij} = -\eta \frac{\delta E(\{w\}|\ldots)}{dw_{ij}} \qquad \eta < 1$$

Serve un'**approssimazione iniziale** per i pesi $W_{ini} = \{w_j\}_{ini}$.

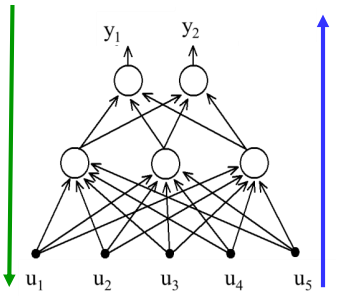
La pratica dell'apprendimento supervisionato

Fino a quando l'apprendimento non è stato completato:

Forward pass;
1. Presentazione di un pattern di input / output (dati).
2. Calcolo dell'output della rete con il pattern corrente (modello).
3. Calcolo di una misura di errore (distanza, discrepanza, entropia).

Backwards pass (cf. backpropagation);
4. Calcolo dei gradienti (apprendimento).
5. Aggiornamento dei pesi (apprendimento).

Aggiornamento dei pesi:
- Per trial (ogni pattern)
- Per epoca (ogni insieme di pattern).



Apprendimento supervisionato tramite gradiente

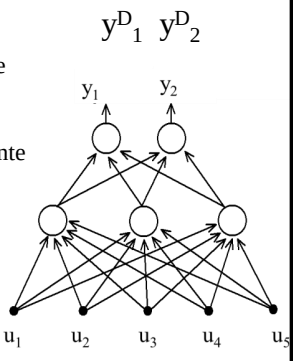
Coppie input/output note.

Definizione di una **funzione costo** che misuri l'errore sull'uscita (abbiamo visto la norma).

Modifica dei valori dei parametri (pesi) in modo tale che la funzione costo sia minimizzata.

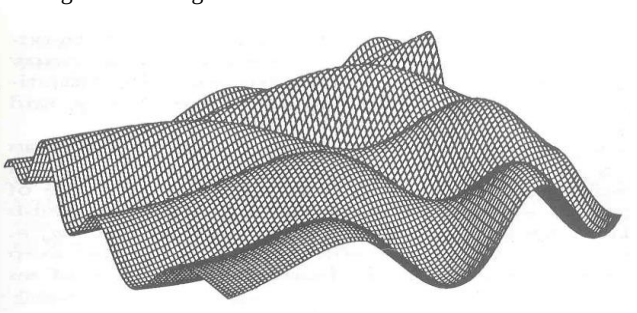
Reti multi-strato hanno elevata capacità computazionale, ma anche elevata complessita'.

La funzione di uscita è ottenuta come funzione di funzioni, gradiente complesso.



Problemi nell'apprendimento supervisionato tramite gradiente

- Nota: W_{ini} è generalmente casuale e può condizionare la convergenza degli algoritmi iterativi. Solitamente si sceglie molto piccolo e random.
- I problemi di convergenza sono legati all'esistenza di minimi locali del funzionale E(w | ….)



Gradiente è basato su una descrizione locale della funzione E(w).
Ci si muove verso la direzione di minimo (locale)

Modelli lineari e non lineari

Classificazione alternativa dei modelli. Vengono utilizzate classi molto diversi di algoritmi per stimare i parametri di questi due tipi di modelli.

$$z(p(x,y)) = f(x) = \sum_i w_i x$$

f(.) è funzione lineare nei $\{w_i\}$

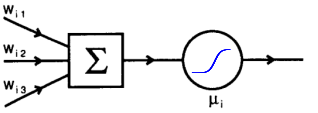
$$z(p(x,y)) = \sum_i f_i(p;w))$$

f(.) è funzione non lineare

e.g. $f(.) = e^{w\,x}$
$f(.) = x \ln(w\,x)$
....

Unità di attivazione lineari: sistema

$$y_j = g\left(\sum_{i=1}^{M} w_{ij}u_i - \mu_j\right) = g\left(\sum_{i=0}^{M}\left(w_{ij}u_i\right)\right)$$



Caso lineare (g(.) = 1):

$$y_j = \sum_{i=1} w_{ij}u_i - \mu_j = \sum_{i=0}\left(w_{ij}u_i\right) \qquad \text{====>} \qquad \mathbf{Y} = \mathbf{W}\,\mathbf{U}$$

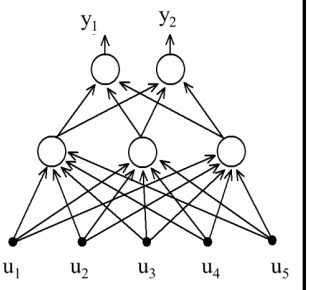
Soluzione di un sistema lineare nei pesi!!

Condizione di risolubilità: **W** di rango massimo →
{w} sono linearmente indipendenti.

Unità lineari, soluzione iterativa

$$J = E(\mathbf{w}) = \frac{1}{2}\sum_p\left[\sum_j\left(y_{jp}^D - y_{jp}\right)^2\right] = \frac{1}{2}\sum_p\left[\sum_j\left(y_{jp}^D - \left(\sum_i w_{ij}u_{ip}\right)\right)^2\right]$$

$$\Delta w_{ij} = -\eta\frac{\partial}{\partial w_{ij}}\frac{1}{2}\sum_j\left(y_j^D - \left(\sum_i w_{ij}u_i\right)\right)^2$$



$$\Delta w_{ij} = -\eta\sum_j\left(y_j^D - \sum_k w_{kj}u_k\right)(-u_i) = +\eta\sum_j\left(y_j^D - y_j\right)u_i$$

Hebbian learning

δ rule (Hoff, 1960)

Apprendimento supervisionato reti lineari

Fino a quando l'apprendimento non è stato completato:

Forward pass;

$y_i = w_1u_1 + w_2u_2 + w_3u_3 + w_0u_0$

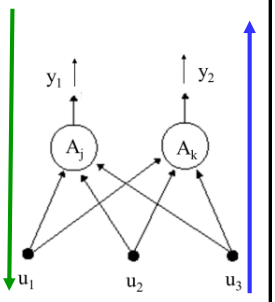
1. Presentazione di un pattern di input / output (dati).
2. Calcolo dell'output della rete con il pattern corrente (modello).
3. Calcolo di una misura di errore (distanza, discrepanza, entropia). $E = (y_i^D - y_i)^2$

Backwards pass;

4. Calcolo dei gradienti (apprendimento). $\frac{\partial E}{dw_{ij}} = (y_i^D - y_i)\, u_j$
5. Aggiornamento dei pesi (apprendimento).

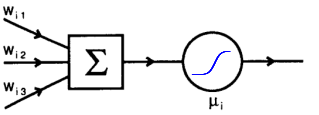
w_{ij} += $-\eta(y_i^D - y_i)(-u_j)$

Aggiornamento dei pesi:
- Per trial (ogni pattern)
- Per epoca (ogni insieme di pattern).



Unità di attivazione non-lineari

$$y_j = g\left(\sum_{i=1}^{M} w_{ij}u_i - \mu_j\right) = g\left(\sum_{i=0}^{M}\left(w_{ij}u_i\right)\right)$$



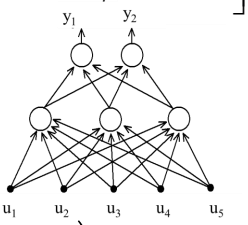
$\sum_{i=0}^{M}\left(w_{ij}u_i\right)$ è l'argomento della funzione di attivazione g(.) che poniamo = z

Apprendimento: minimizzazione dell'errore, $E(w \mid y^D, x)$, sui pattern tra l'uscita desiderata prescritta e quella fornita dal modello.

Unità non-lineari, soluzione iterativa

$$J = E\left(y^D, x^D \mid \mathbf{w}\right) = \frac{1}{2}\sum_p \left[\sum_j \left(y_{jp}^D - y_{jp}\right)^2\right] = \frac{1}{2}\sum_p \left[\sum_j \left(y_{jp}^D - g(\sum_i w_{ij}u_{ip})\right)^2\right]$$

$$\Delta w_{ijp} = -\eta \frac{\partial}{\partial w_{ij}} \frac{1}{2}\sum_j \left(y_{jp}^D - g(\sum_i w_{ij}u_{ip})\right)^2 =$$



$$\Delta w_{ij} = -\eta \sum_j \left(y_{jp}^D - g\left(\sum_k w_{kj}u_{kp}\right)\right) g'\left(\sum_k w_{kj}u_{kp}\right)(-u_i) =$$

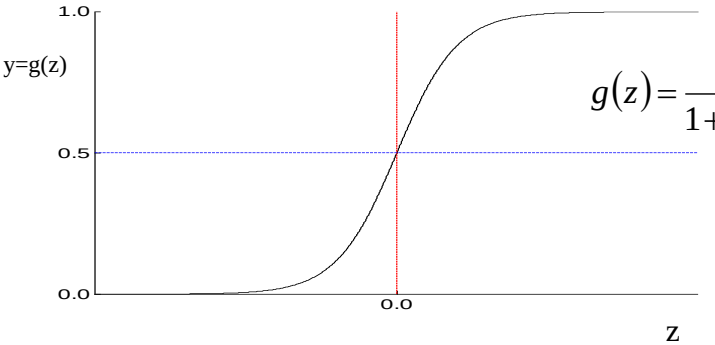
$$+\eta \sum_j \left(y_j^D - y_j\right) g'\left(\sum_k w_{kj}u_{kp}\right) u_i$$

δ rule

Perceptrone con unità di attivazione logistiche

$$y_j = g\left(\sum_i w_{ij}u_i\right) \qquad z = \sum_i \left(w_{ij}u_i\right)$$

Uscita compresa tra 0 e 1



$$g(z) = \frac{1}{1+e^{-z}}$$

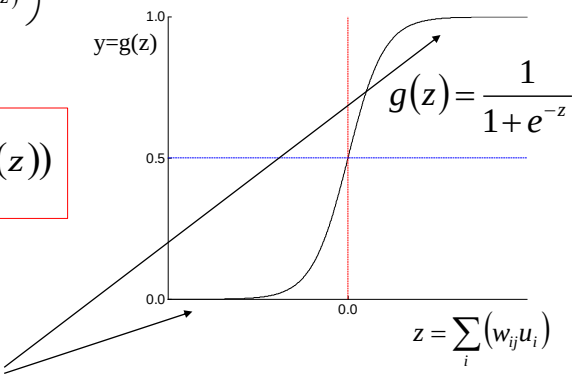
Perceptrone con unità di attivazione logistiche

$$g'(z) = \frac{e^{(-z)}}{\left(1+e^{(-z)}\right)^2} =$$

$$= \frac{1}{1+e^{(-z)}}\left(1-\frac{1}{1+e^{(-z)}}\right) =$$

$$g'(z) = g\ (z)\cdot(1-g(z))$$

Uscita compresa tra 0 e 1



Saturazione
(cf. dead units)

Update dei pesi per funzione logistica

$$J = E(\mathbf{w}) = \frac{1}{2}\sum_p \left[\sum_j \left(y_{jp}^D - y_{jp}\right)^2 = \frac{1}{2}\sum_j \left(y_{jp}^D - g(\sum_i w_{ij}u_{ip})\right)^2\right]$$

$$\Delta w_{ij} = +\eta \sum_j \left(y_{jp}^D - g\left(\sum_k w_{kj}u_{kp}\right)\right) g'\left(\sum_k w_{kj}u_{kp}\right) u_i$$

$y_j = g(z) = \frac{1}{1+e^{-z}}$ $\qquad$ $g'(z) = g(z)(1-g(z))$

$$\Delta w_{ijp} = +\eta \sum_j \left(y_{jp}^D - g(.)\right) g'(.) u_i = +\eta\left(y_{jp}^D - y_j\right) u_{ip} y_j (1-y_j)$$

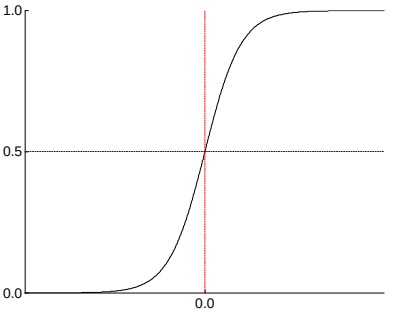
Update dei pesi per funzione logistica

$$\Delta w_{ijp} = +\eta \sum_j \left(y_{jp}^D - g(.)\right) g'(.) u_i = +\eta \left(y_{jp}^D - y_j\right) u_{ip} y_j (1 - y_j)$$

NB y_i =[0 1].

Per $y_i = 0$ o $y_i = 1$ non c'è apprendimento anche se l'uscita è sbagliata. Quando si verifica questa situazione?

Si cerca di mantenere le unità lontane della saturazione.



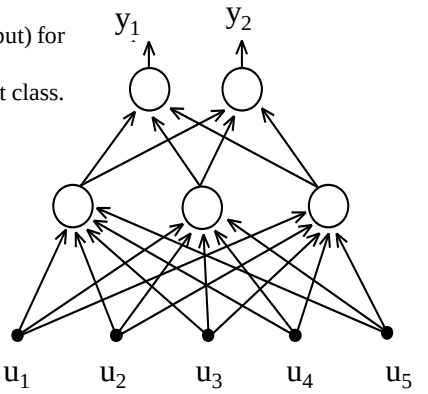
Funzioni costo

$\|Y^D - g(\mathbf{W}\,\mathbf{U})\|^2$ Squared loss function. Gaussian error.

For classification
y_i is the predicted probability value (model output) for class i and
y'_i is the true probability (desired output) for that class.

$H_{y'}(y) = -\sum_i y'_i \log(y_i)$ cross-entropy loss

y_i bounded between (0 1]
y_i is maximum for $y_i = ½$



Sommario

Dal neurone artificiale alle reti neurali

L'apprendimento in reti di perceptroni

Esempio con unità lineari

Esempio apprendimento modello lineare

$$J = E(\mathbf{w}) = \frac{1}{2}\sum_p \left[\sum_j \left(y_{jp}^D - y_{jp} \right)^2 \right] = \frac{1}{2}\sum_p \left[\sum_j \left(y_{jp}^D - \left(\sum_i w_{ij} u_{ip} \right) \right)^2 \right]$$

$$\Delta w_{ij} = -\eta \frac{\partial}{\partial w_{ij}} \frac{1}{2} \sum_j \left(y_j^D - \left(\sum_i w_{ij} u_i \right) \right)^2$$

$$\Delta w_{ij} = +\eta \sum_j \left(y_j^D - \left(\sum_j w_{ij} u_i \right) \right) u_i = +\eta \sum_j \left(y_j^D - y_j \right) u_i$$

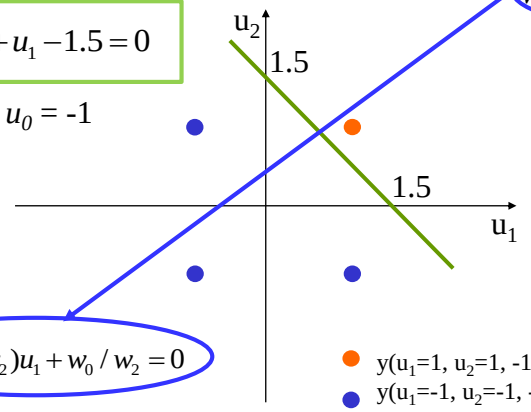
Hebbian learning

δ rule (Hoff, 1960)

Esempio – AND logico

$u_2 + u_1 - 1.5 = 0$

$u_0 = -1$



$w_0 u_0 + w_1 u_1 + w_2 u_2 = 0$

In verde la retta $\mathbf{w'} \cdot \mathbf{u} = 0$ che taglia il piano $u_1 u_2$.

$u_0 = -1$
$w_0 = 1.5$

$w_1 = 1$
$w_2 = 1$

$u_2 + (w_1 / w_2) u_1 + w_0 / w_2 = 0$

$y(u_1=1, u_2=1, -1) = 1$
$y(u_1=-1, u_2=-1, -1) = -1$

$u_2 + u_1 - 1.5 = 0$

$\Downarrow$

$w_1 / w_2 = 1 \quad w_0 / w_2 = -1.5$

$\Rightarrow$

Esistono più soluzioni
Separabilità lineare.

$w_2 = k \quad w_1 = k \quad w_0 = 1.5 * k$

Ottimizzazione dei {w}

$w_0 u_0 + w_1 u_1 + w_2 u_2 = 0$
$u_0 = -1 \Rightarrow w_0$

Matrice dei termini noti: $b = y^D$

$$b = \begin{bmatrix} -1 \\ -1 \\ -1 \\ +1 \end{bmatrix}$$

4 equazioni
3 incognite

u_1	u_2	y	y^D
-1	-1	-1	-1
-1	1	+1	-1
1	-1	-1	-1
1	1	-1	+1

Matrice dei coefficienti: A

$$A = \begin{bmatrix} -1 & -1 & -1 \\ -1 & +1 & -1 \\ -1 & -1 & +1 \\ -1 & 1 & 1 \end{bmatrix}$$

Coeff Soglia ← u_0 u_2 u_1

Vettore delle incognite: x = w

Ax = b

$x = (A'*A)^{-1}*A'*b$ ➔ $w = \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \end{bmatrix}$

Soluzione ottima: minimizzo implicitamente la distanza tra la retta e i 4 punti.

Verifica

$w_0u_0+w_1u_1+w_2u_2 = 0$

$u_0 = -1 \Rightarrow w_0$

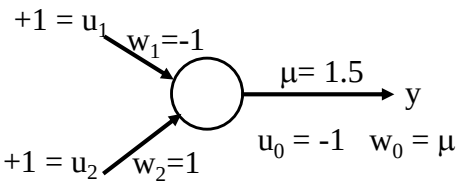
$$w = \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \end{bmatrix}$$

+0.5(-1) + 0.5(-1) + 0.5(-1) = -1.5
+0.5(-1) + 0.5(-1) + 0.5(+1) = -0.5
+0.5(-1) + 0.5(+1) + 0.5(-1) = -0.5
+0.5(-1) + 0.5(+1) + 0.5(+1) = 0.5

u_1	u_2	y	y^D
-1	-1	-1	-1
-1	1	+1	-1
1	-1	-1	-1
1	1	-1	+1

Calcolo dell'uscita per un set di {w}

Inizializzo i pesi: $w_1 = -1$, $w_2 = 1$, $w_0 = 1.5$



u_1	u_2	y	y^D
-1	-1		-1
-1	1		-1
1	-1		-1
1	1	-1.5	+1

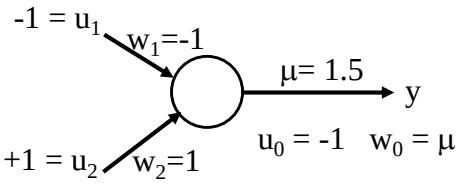
Pattern: U=|1, 1| -> $y^D = +1$

$$y = \sum_{i=1} w_i u_i - \mu = \sum_{i=0} (w_i u_i) \quad = (-1)(1) + (1)(1) + (1.5)(-1) = -1.5 << +1$$

Questi pesi non vanno bene. Come si può procedere?

Soluzione iterativa
Delta rule I: calcolo dell'uscita

Inizializzo i pesi: $w_1 = -1$, $w_2 = 1$, $w_0 = 1.5$



u_1	u_2	y	y^D
-1	-1		-1
-1	1	0,5	-1
1	-1		-1
1	1		+1

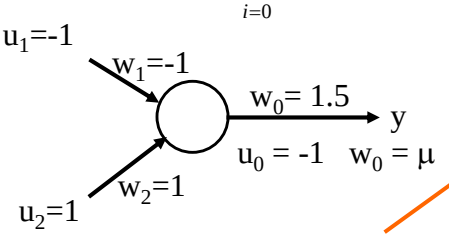
Pattern: U=|-1, 1| -> $y^D = -1$

$$y = \sum_{i=1} w_i u_i - \mu = \sum_{i=0} (w_i u_i) = (-1)(-1) + (1)(1) + (-1.5)(+1) = 0.5 >> -1$$

Questi pesi non vanno bene neppure per questo pattern. Come si può procedere?

Delta rule II: Calcolo dell'errore

$$y = \sum_{i=0} (w_i u_i) = 0.5$$



u_1	u_2	y	y^D
-1	-1		-1
-1	1	0,5	-1
1	-1		-1
1	1		+1

Pattern: U=|-1, 1| -> $y^D = -1$

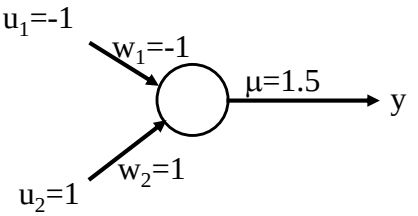
Errore = $1/2(y^D - y)^2 = 1/2(-1 - (0.5))^2 = 0.5*(-1.5^2) = 2.25$

Delta rule III: calcolo del gradiente

$$\frac{dErrore}{dw_1} = \left(y_i^D - y_i\right)u_1 = (-1-0.5)(-1) = 1.50$$

$$\frac{dErrore}{dw_2} = \left(y_i^D - y_i\right)u_2 = (-1-0.5)(+1) = -1.50$$

$$\frac{dErrore}{d\mu} = -\frac{dErrore}{dw_0} = +1.50$$



Delta rule IV: aggiornamento pesi

$$\Delta w_{ij} = +\eta\left(y_j^D - y_j\right)u_i$$

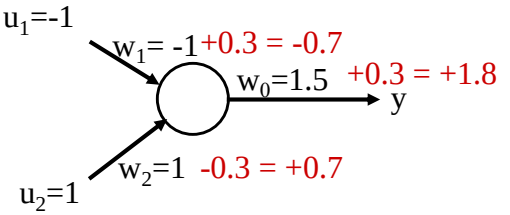
U=|-1, 1| y^D = -1

η = 0.2

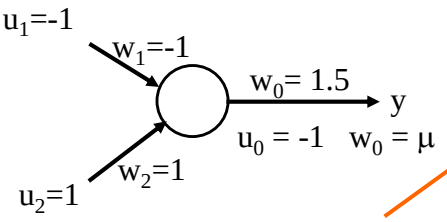
$$\Delta w_1 = \eta\left(y_i^D - y_i\right)u_1 = \eta(-1-0.5)(-1) = +0.30$$

$$\Delta w_2 = \eta\left(y_i^D - y_i\right)u_2 = \eta(-1-0.5)(+1) = -0.30$$

$$\Delta w_0 = \eta\left(y_i^D - y_i\right)u_0 = \eta(-1-0.5)(-1) = +0.30$$



Delta rule V: Nuovo valore di uscita

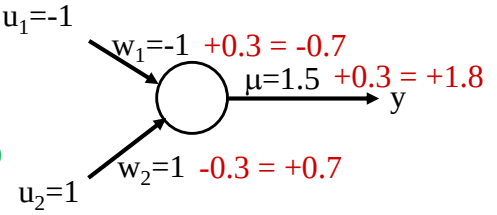


u_1	u_2	y	y^D
-1	-1		-1
-1	1	-0.4	-1
1	-1		-1
1	1		+1

Pattern: U=|-1, 1| -> $y^D = -1$

$y = \sum_{i=0} (w_i u_i) =$ -0.4 > -1 ma OK

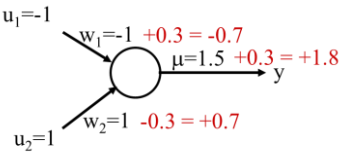
Potrei fermarmi qui (per questo pattern)



Risultato finale

$w_0u_0+w_1u_1+w_2u_2 = 0$

$u_0 = -1 \Rightarrow w_0$



u_1	u_2	y	y^D
-1	-1	**-1.8**	-1
-1	1	-0.4	-1
1	-1	-3.2	-1
1	1	**-1.8**	+1

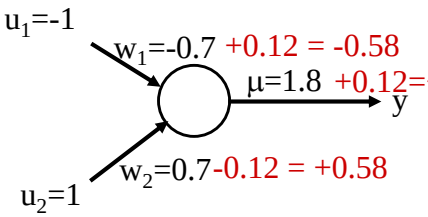
-0.7x(-1) + 0.7x(-1) + 1.8x(-1) = -1.5
-0.7x(-1) + 0.7x(+1) + 1.8x(-1) = -0.5
-0.7x(+1) + 0.7x(-1) + 1.8x(-1) = -0.5
-0.7x(+1) + 0.7x(+1) + 1.8x(-1) = 0.5

Predico correttamente 3 pattern su 4, è un apprendimento polarizzato, specializzato sul pattern [-1 1; -1]
-> **devo fare ruotare tutti i pattern**.

Delta rule VI- Nuovo aggiornamento

U=|-1, 1| $y^D = -1$

$\eta = 0.2$



$$y = \sum_{i=1} w_i u_i - \mu = \sum_{i=0} (w_i u_i) = \textbf{-0.4}$$

-0.76

$$\Delta w_{ij} = +\eta \left(y_i^D - y_i \right) u_j$$

$$\Delta w_1 = \eta \left(y_i^D - y_i \right) u_1 = \eta(-1-0.4)(-1) = +0.12$$

$$\Delta w_2 = \eta \left(y_i^D - y_i \right) u_2 = \eta(-1-0.4)(+1) = -0.12$$

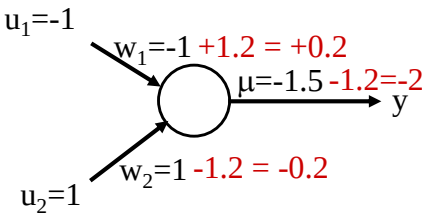
$$\Delta w_0 = \eta \left(y_i^D - y_i \right) u_0 = \eta(-1-0.4)(-1) = +0.12$$

Che relazione c'è tra i pesi e la retta che separa le uscite positive da quelle negative?

Cattiva scelta di η

U=|-1, 1| $y^D = -1$

$\eta = 0.8$



$$y = \sum_{i=1} w_i u_i - \mu = \sum_{i=0} (w_i u_i) = \textbf{-3.1}$$

$u_0 = -1 \quad w_0 = \mu$

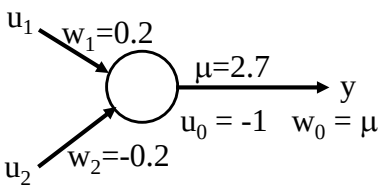
$$\Delta w_{ij} = +\eta \left(y_j^D - y_j \right) u_i$$

$$\Delta w_1 = \eta \left(y_i^D - y_i \right) u_1 = \eta(-1-0.5)(-1) = +1.2$$

$$\Delta w_2 = \eta \left(y_i^D - y_i \right) u_2 = \eta(-1-0.5)(+1) = -1.2$$

$$\Delta w_0 = \eta \left(y_i^D - y_i \right) u_0 = \eta(-1-0.5)(-1) = +1.2$$

Risultato con cattiva scelta di η



	u_1	u_2	y^D	
	-1	-1	-1	a
	-1	1	-1	b
	1	-1	-1	c
	1	1	1	d

a $y = \sum_{i=1} w_i u_i - \mu = \sum_{i=0} (w_i u_i)$ = (0.2)(-1) + (-0.2)(1) – 2.7 = -3.1

b $y = \sum_{i=1} w_i u_i - \mu = \sum_{i=0} (w_i u_i)$ = (0.2)(-1) + (-0.2)(1) – 2.7 = -2.9

c $y = \sum_{i=1} w_i u_i - \mu = \sum_{i=0} (w_i u_i)$ = (0.2)(1) + (-0.2)(-1) – 2.7 = -2.3

d $y = \sum_{i=1} w_i u_i - \mu = \sum_{i=0} (w_i u_i)$ = (0.2)(1) + (-0.2)(1) – 2.7 = **-2.7**

Errato su d. Specializzazione su a, b, c

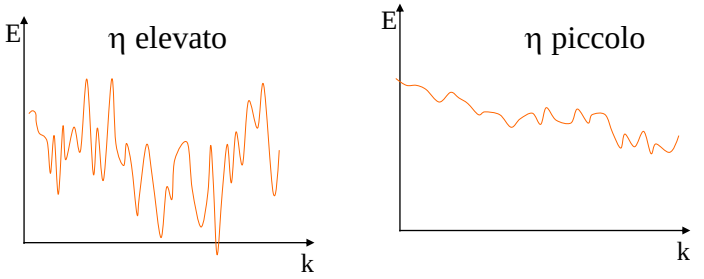
Ruolo di η – learning rate

$$\Delta w_{ij} = +\eta \left(y_j^D - y_j\right) u_i$$

η = 0.2 è un valore adeguato?
η = 0.8 è un valore adeguato?

Calmiera il Δw_{ij} per evitare che :

- Un peso sia specifico di un'unità ingresso-uscita.
- Oscillazioni durante l'apprendimento senza convergenza.



η può variare durante l'addestramento.

Riassunto – topologia

I neuroni connessionisti sono basati su:
•Ricevere una somma pesata degli ingressi.
•Trasformarla secondo una funzione non-lineare (scalino o logistica)
•Inviare il risultato di questa funzione all'uscita o ad altre unita'.

Le reti neurali sono topologie ottenute connettendo tra loro i neuroni in modo opportuno e riescono a calcolare funzioni molto complesse.

Riassunto - Apprendimento

Algoritmi iterativi per adattare il valore dei parametri (pesi).

Definizione di una funzione costo che misura la differenza tra valore fornito e quello desiderato.

Algoritmo (gradiente) che consente di aggiornare i pesi in modo da minimizzare la funzione costo.

Training per pattern (specializzazione) o per epoche.

Direzioni di sviluppo della ricerca

1) Spiking neurons models – Modelli computazionali a neurone singolo. Evoluzione temporale (software: Neuron, Genesis)

2) Continous activation models – Apprendimento di pattern.

Deep learning is a branch of machine learning based on a set of algorithms that attempt to model high-level abstractions in data by using multiple processing layers, or otherwise composed of multiple non-linear transformations (e.g. tensor flow in Keras https://keras.io/)

Capsule networks. Modules connected.

Transformers (Pre-trained DNN)

Sommario

Dal neurone artificiale alle reti neurali

L'apprendimento in reti di perceptroni

Esempio con unità lineari ed accenno ad unità non-lineari