# Classificatori 2 (approccio connessionistico)

Neuroni biologici e neuroni artificiali

Threshold Logic Unit

## Come si costruisce il classificatore ?

- Appurate le caratteristiche che dovrebbe esibire un classificatore efficiente, si pone il problema della costruzione del classificatore.
- **E' possibile una soluzione algoritmica ?**
  E', cioè possibile, definire un algoritmo per classificare caratteri, parlato, immagini,… ?
- Dopo 40 anni di sforzi in questo senso la risposta è chiaramente negativa.
- Perché ?
- Ci sono alternative ?

# L'apprendimento

- L'unica alternativa percorribile è quella di apperendere a risolvere i problemi a partire da esempi (*learning by examples*).
- *Apprendimento* (*learning*): ogni metodo che, nella costruzione di un classificatore, combina informazioni empiriche provenienti dall'ambiente e conoscenza a priori del contesto del problema.
- Le informazioni empiriche sono di solito nella forma vista di campioni di esempio (training set).
- Conoscenza a priori: invarianti, correlazioni, …

# Paradigmi di apprendimento

Esistono diversi paradigmi di apprendimento:

- Apprendimento *supervisionato* (*supervised*):

  per ogni campione del training set è provvista la classe di appartenenza. Obiettivo dell'apprendimento è quello di minimizzare gli errori (o il costo di classificazione).

- Apprendimento *non supervisionato* (*unsupervised*):

  non sono fornite esplicite informazioni sulla classe dei campioni del training set. Obiettivo dell'apprendimento è quello di formare dei *raggruppamenti* (*clusters*) dei campioni generalmente sulla base di una distanza. Spesso è definito dall'esterno il numero dei clusters da produrre. Questo paradigma viene definito anche *clustering*.

## Origini delle reti neurali

- La storia delle reti neurali si innesta in quella dell'Intelligenza Artificiale.
- Obiettivo dell'IA:

  ***replicare per mezzo di macchine quanto più possibile dell'attività mentale umana***

- Tesi forte dell'IA:

  *L'attività mentale risulta dall'esecuzione di qualche sequenza ben definita di operazioni di manipolazione di simboli (algoritmo) .*

## Ci vuole un po' di cervello…

Ogni tentativo di replicazione dell'attività mentale deve fare i conti con il cervello umano.

**The brain –**

**that's my second most favourite organ!**

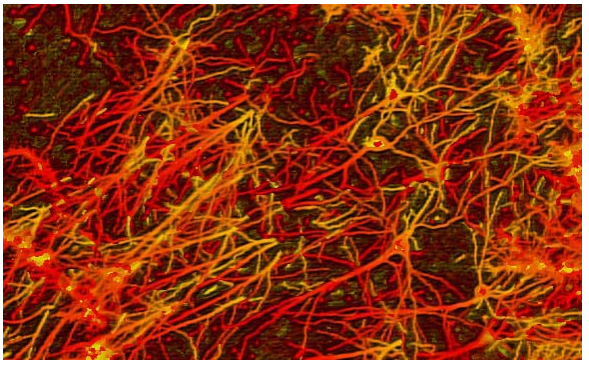
Woody Allen

# Confronto cervello-calcolatore

| | # Elementi di calcolo | Dimensione el. di calcolo | Velocità di elaborazione | Tipo di elaborazione | Tolleranza ai guasti | Capacità di apprendimento | Intelligenza coscienza |
|---|---|---|---|---|---|---|---|
|  | $10^{14}$ **sinapsi** | $10^{-6}$m | 100 Hz | **Parallela, distribuita** | si | si | si |
|  | $10^{8}$ **transistor** | $10^{-6}$m | 1 GHz | **Seriale, centralizzata** | no | poca | no |

# Reti neurali biologiche

Gli elementi fondamentali del sistema nervoso centrale sono i **neuroni** (presenti in misura di circa $10^{11}$).

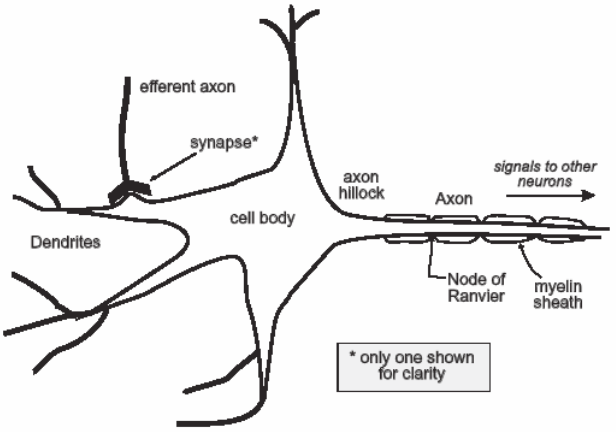


# Il neurone biologico

Un neurone è composto di tre parti: il *corpo cellulare* (o *soma*), le *dendriti* e l'*assone* (o *cilindrasse*).

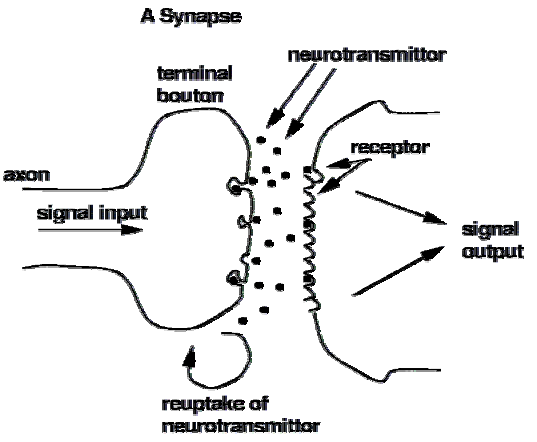
La comunicazione tra neuroni avviene tramite segnali elettrici che sono trasmessi lungo l'assone.

Questo si connette ad una dendrite di un altro neurone tramite un terminale detto *sinapsi*.




# Attivazione del neurone

- L'impulso arrivato alla sinapsi stimola il rilascio di neurotrasmettitori
- Questi, captati dai ricettori posti sulla dendrite, inducono una modifica nel potenziale della membrana dendritica (PSP)
- Il PSP può iperpolarizzare o depolarizzare la membrana dendritica, tendendo a inibire o a eccitare la generazione di nuovi impulsi.

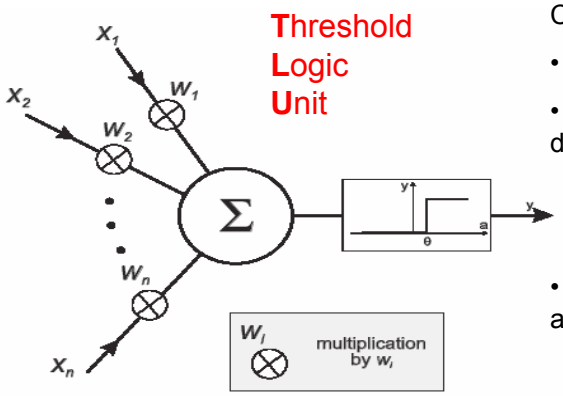


# Attivazione del neurone

- Ogni PSP attraversa la dendrite su cui è stato generato e si diffonde sul soma, raggiungendo la base dell'assone. Il neurone integra gli effetti delle migliaia di PSP presenti sulle sue dendriti
- Nel caso superi una soglia, genera un impulso che inizia a propagarsi lungo l'assone, così da ricominciare l'intero processo in un nuovo neurone.

# Il modello di neurone artificiale di McCulloch-Pitts (1943)

**TLU**:
**T**hreshold
**L**ogic
**U**nit




Caratteristiche:

• I segnali sono binari ($x_i$=0/1)

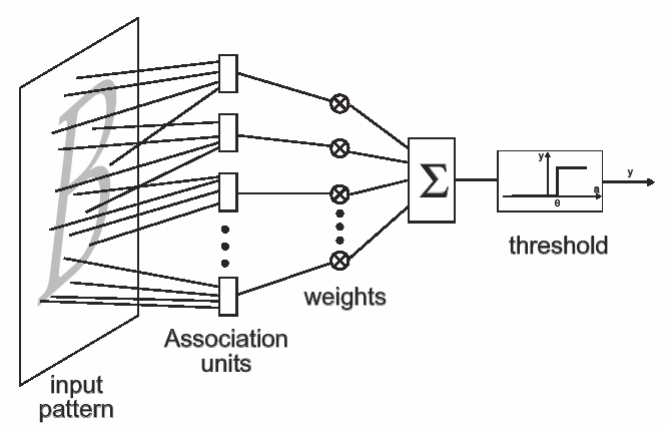
• La funzione di attivazione è definita come:

$$a = \sum_{i=1}^{n} w_i x_i$$

• L'uscita è definita in base alla regola:

$$y = \begin{cases} 1 & if \quad a \geq \vartheta \\ 0 & if \quad a < \vartheta \end{cases}$$

# Il *Perceptron* (Rosenblatt, 1962)



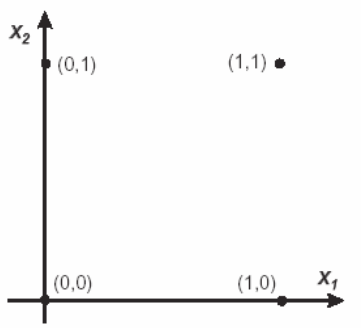

# Che cosa può fare una TLU?

- Consideriamo due ingressi e valori dei pesi: $w_1=1$, $w_2=1$, $\vartheta=1.5$
- La relazione ingresso-uscita della TLU sarà:

| $x_1$ | $x_2$ | *a* | *y* |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 2 | 1 |

Questo comportamento può essere interpretato come una *classificazione* dei campioni $(x_1,x_2)$ in ingresso, ai quali vengono fatte corrispondere due classi, identificate da y=0 e da y=1.

# Interpretazione geometrica del comportamento della TLU




E' possibile rappresentare i campioni in ingresso in uno spazio 2D (spazio dei pattern) avente come assi i valori $x_1$ e $x_2$ in ingresso alla TLU.

In generale, per *n* ingressi, lo spazio dei pattern sarà *n*D.

# Interpretazione geometrica del comportamento della TLU

- Data la relazione ingresso-uscita della TLU, lo spazio dei pattern sarà diviso in due sottospazi:

$$S_0 = \{(x_1, x_2) : w_1 x_1 + w_2 x_2 < \theta\}$$

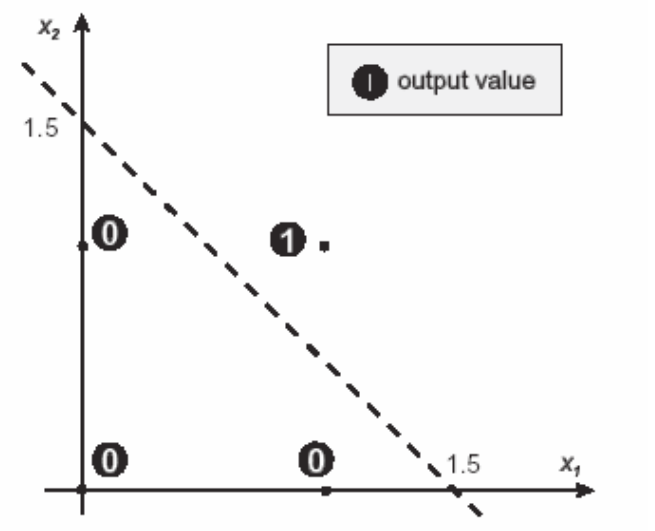
$$S_1 = \{(x_1, x_2) : w_1 x_1 + w_2 x_2 \geq \theta\}$$

separati dalla retta:

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

che costituisce la frontiera (*decision boundary*) tra le due regioni.

# Interpretazione geometrica del comportamento della TLU




# Addestramento della TLU

- La fase di addestramento consiste nel modificare i pesi e la soglia della TLU in modo che questa "si comporti secondo quanto desiderato"
- Per questo scopo si usa un insieme (*training set*) formato da campioni affiancati dall'uscita desiderata.

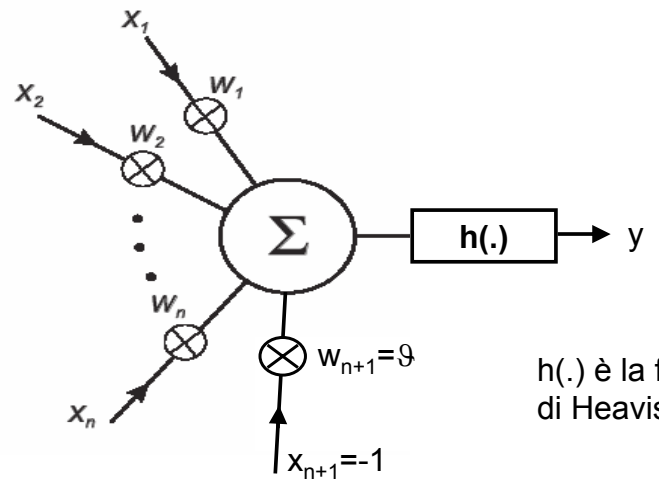
$$TS = \left\{ (\mathbf{x}_j, y_j) \right\}_{j=1,\ldots,M}$$

# Addestramento della TLU

L'uscita della TLU è:

$$y = \begin{cases} 1 \;\; if \;\; \sum_{i=1}^{n} w_i x_i \geq \vartheta \\ 0 \;\; if \;\; \sum_{i=1}^{n} w_i x_i < \vartheta \end{cases} \quad \text{equivalente a} \quad y = \begin{cases} 1 \;\; if \;\; \sum_{i=1}^{n} w_i x_i - \vartheta \geq 0 \\ 0 \;\; if \;\; \sum_{i=1}^{n} w_i x_i - \vartheta < 0 \end{cases}$$

Per comodità di calcolo, possiamo considerare la TLU con un ingresso in più $x_{n+1}$ il cui valore è fisso a -1, mentre il peso corrispondente $w_{n+1}$ è uguale a $\vartheta$.




h(.) è la funzione di Heaviside

# Addestramento della TLU

- Se definiamo i vettori:

  $\mathbf{w} = (w_1, w_2, \ldots, w_n, \vartheta)$ e $\mathbf{x} = (x_1, x_2, \ldots, x_n, -1)$

  lo scopo dell'addestramento è modificare **w** in modo che $h(\mathbf{w} \cdot \mathbf{x}_j) = y_j$ per ogni punto del training set.
- Possiamo non considerare la non linearità h(.) ponendo come obiettivo dell'addestramento:

  $\mathbf{w} \cdot (\mathbf{x}_j\ t_j) \geq 0$ per ogni punto del training set

  dove $t_j$=-1 se $y_j$=0 e $t_j$=+1 se $y_j$=1.

# Addestramento della TLU: definizione di errore

Per una certa configurazione dei pesi, possiamo definire come errore di classificazione della TLU la quantità:

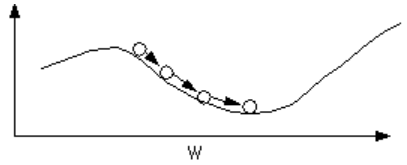
$$E_{TLU}(\mathbf{w}) = -\sum_{j \in J_M} \mathbf{w} \cdot (\mathbf{x}_j t_j)$$

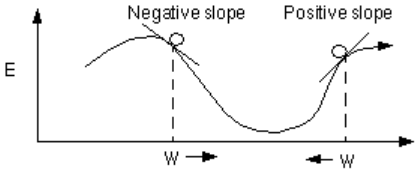
dove $J_M = \{ j : \mathbf{w} \cdot (\mathbf{x}_j t_j) < 0 \}$ indica l'insieme dei punti per cui la TLU compie un errore di classificazione.

E' allora necessario definire un algoritmo che minimizzi l'errore sul TS

# Algoritmo di discesa lungo il gradiente (*gradient descent*)

Per raggiungere il punto di minimo dell'errore, a partire da una configurazione iniziale dei pesi, possiamo muovere questi nella direzione in cui l'errore decresce più rapidamente.




Possiamo quindi porre la regola di modifica dei pesi:

$\mathbf{w}^{k+1} = \mathbf{w}^{k} - \eta \nabla_{w} E_{TLU}$

dove $\eta$ è detto *learning rate*

# Algoritmo di discesa lungo il gradiente (*gradient descent*)

Per il singolo $w_i$, l'equazione di aggiornamento alla k-ma iterazione diventa:

$$w_i^{k+1} = w_i^{k} - \eta \left. \frac{\partial E_{TLU}}{\partial w_i} \right|_{\mathbf{w}^k}$$

che, applicata alla definizione di errore per ogni campione appartenente a $J_M$, diventa:

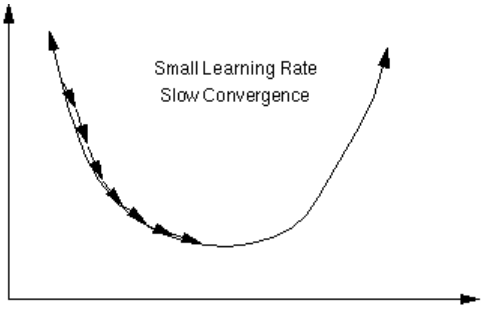
$$w_i^{k+1} = w_i^{k} + \eta x_{ji} t_j$$

# Algoritmo di discesa lungo il gradiente (*gradient descent*)

```
repeat
  for j=1,M
    valuta l'uscita y della TLU per (x_j, y_j)
    if y<>y_j then
      for i=1,n+1
        w_i=w_i+ηx_ji t_j
      end for
    end if
  end for
until non ci sono errori
```

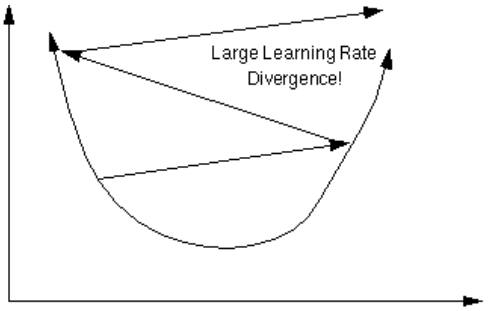
# Come scegliere il learning rate ?

- Il valore di $\eta$ influisce sulla rapidità di convergenza dell'algoritmo, per cui un valore basso può risultare in una lentezza eccessiva



# Come scegliere il learning rate ?

- Di contro, un valore troppo alto può far divergere l'algoritmo

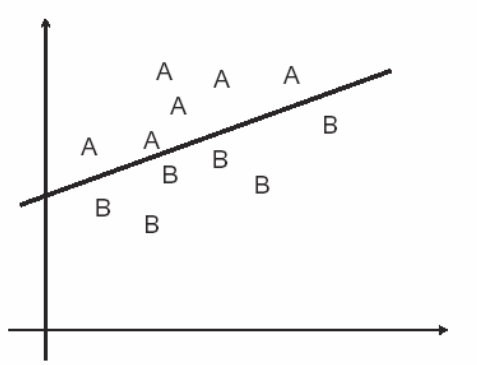



# Reti di neuroni artificiali

# 2 classi linearmente separabili

Con una TLU è possibile risolvere i problemi in cui le classi siano linearmente separabili.

E se le classi sono più di 2?



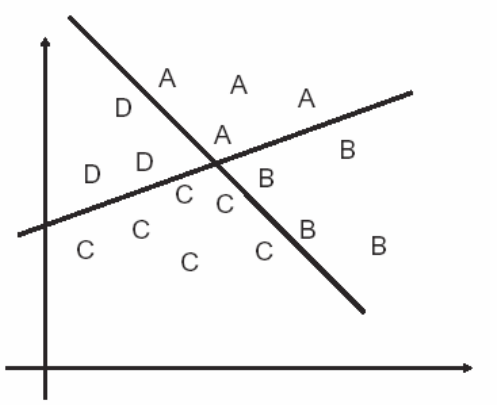

# 4 classi linearmente separabili

In questo problema le 4 classi sono separabili a coppie.

(A,B) - (C,D)

(A,D) - (B,C)

E' possibile costruire un classificatore basato su TLU ?



# 4 classi linearmente separabili

Possiamo utilizzare due variabili binarie per codificare le due partizioni.

| | 1 | 0 |
|---|---|---|
| $y_1$ | (A B) | (C D) |
| $y_2$ | (A D) | (B C) |

Sulla base dei valori che queste assumono, è possibile risalire alla singola classe.

| $y_1$ | $y_2$ | Class |
|---|---|---|
| 0 | 0 | C |
| 0 | 1 | D |
| 1 | 0 | B |
| 1 | 1 | A |

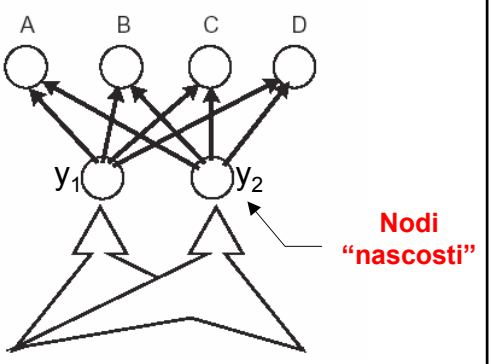
# 4 classi linearmente separabili

Per realizzare il classificatore, bisogna addestrare due TLU che, in fase operativa, dovranno essere completate con altre unità che realizzano opportune funzioni logiche.

Es.:

A = $y_1$ and $y_2$

B = $y_1$ and not $y_2$



# Informazioni necessarie

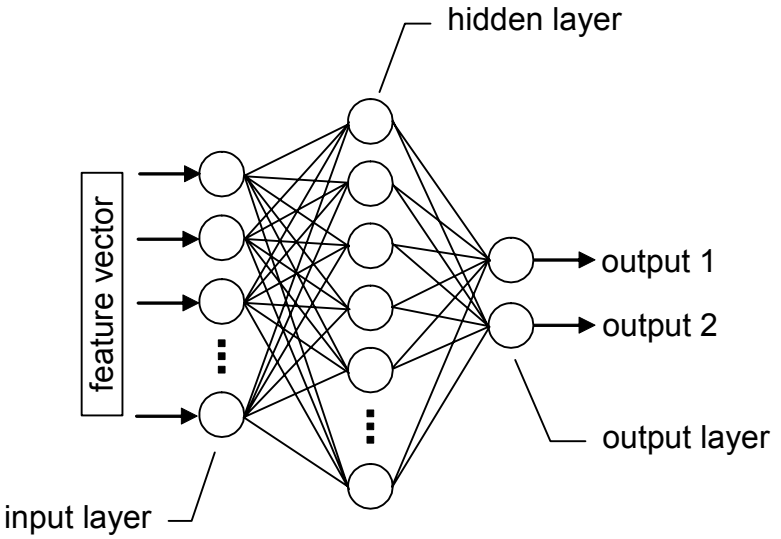
**Possiamo costruire una rete di TLU che risolve il problema. Da notare:**

- E' necessario dapprima addestrare le TLU e quindi definire i pesi delle TLU del livello di uscita che realizzano le funzioni logiche
- Il raggruppamento (A,C) (D,B) non avrebbe funzionato
- Altre disposizioni delle classi nello spazio delle features avrebbe richiesto una differente organizzazione del classificatore
- Due tipi di informazione necessari per addestrare le due TLU:
  1. Le 4 classi possono essere separate da iperpiani
  2. AB è linearmente separabile da CD così come AD da BC

# Esiste un modo automatico ?

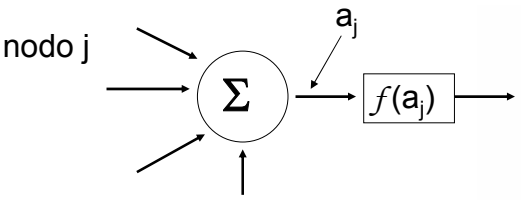
- E' possibile realizzare un algoritmo che si occupi *in toto* della costruzione della rete ?
- Tale algoritmo dovrebbe fissare i pesi delle TLU appartenenti ai due livelli
- Possiamo adattare l'algoritmo visto per una sola TLU ?
- Quali sono i problemi ?
  - Definire la funzione di errore
  - Relazione tra l'uscita ed i pesi
  - Rete su più livelli

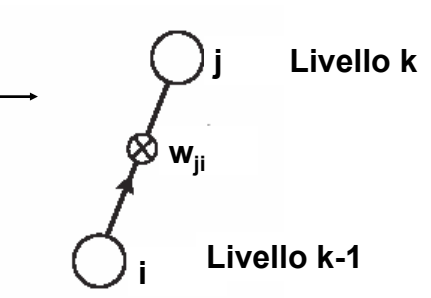
# Struttura della rete
# (feed forward compl. connessa)




# Definizione della rete

Ogni nodo è strutturato come:

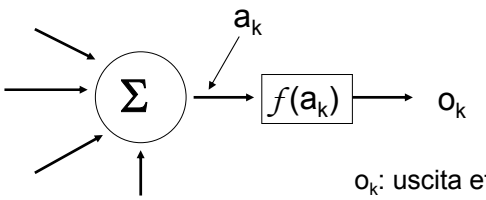



Per il momento, non facciamo ipotesi su $f(.)$

# Definizione della rete

nodo di output




$o_k$: uscita effettiva del nodo di out. k-mo

$d_k$: uscita desiderata del nodo di out. k-mo

Sindrome di errore su k: $|d_k - o_k|$

# Definizione della funzione di errore

- Possiamo far riferimento alla somma dell'errore quadratico rilevato sul livello di uscita:

$$E = \frac{1}{2}\sum_k (d_k - o_k)^2$$

- L'errore misura lo scostamento tra output desiderato e output effettivo
- E' facilmente differenziabile
- E' adatto per molte applicazioni

# L'algoritmo di training

- Cerchiamo di applicare l'algoritmo a gradiente discendente.
- Anche qui sarà valida l'equazione di aggiornamento dei pesi:

$$w_{ji}^{n+1} = w_{ji}^{n} - \eta \left. \frac{\partial E}{\partial w_{ji}} \right|_{\mathbf{w}^n}$$

- Che cosa cambia rispetto al caso della TLU ?

# Calcoliamo il gradiente

- Per la *chain rule:*

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial a_k} \frac{\partial a_k}{\partial w_{kj}}$$

- Calcoliamo i singoli termini:

$$\frac{\partial E}{\partial o_k} = -(d_k - o_k)$$

$$\frac{\partial o_k}{\partial a_k} = \frac{\partial f(a_k)}{\partial a_k} = f'(a_k)$$

$$\frac{\partial a_k}{\partial w_{kj}} = \frac{\partial}{\partial w_{kj}} \sum_j w_{kj} x_j = x_j \quad ?$$

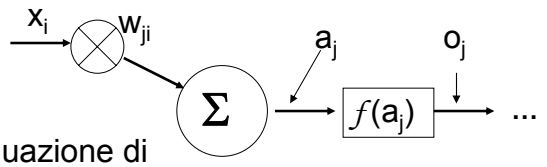
# L'equazione di aggiornamento

- L'equazione di aggiornamento è allora:

$$w_{kj}^{n+1} = w_{kj}^{n} + \eta(d_k - o_k) f'(a_k) x_j \Big|_{\mathbf{w}^n}$$

- **Achtung!**
  Questa regola è applicabile solo ai neuroni del livello di output. Perché ?
- Come si aggiornano i pesi degli altri neuroni ?

# L'equazione di aggiornamento per i neuroni interni




Per valutare l'equazione di aggiornamento:

$$w_{ji}^{n+1} = w_{ji}^{n} - \eta \frac{\partial E}{\partial w_{ji}} \Big|_{\mathbf{w}^n}$$

dobbiamo calcolare il gradiente: $\dfrac{\partial E}{\partial w_{ji}}$

che possiamo scrivere:

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}}$$

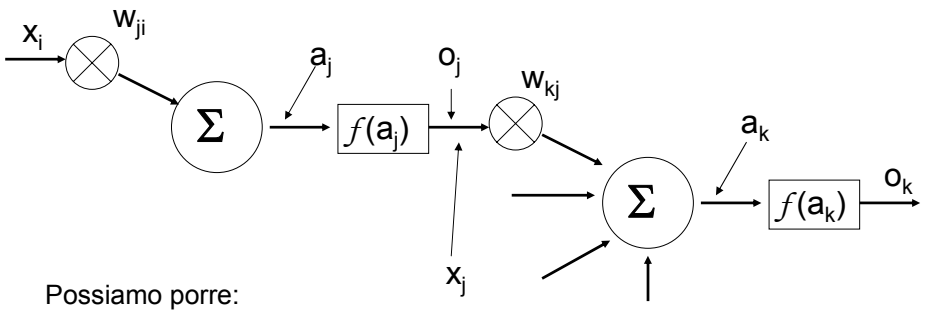
# L'equazione di aggiornamento per i neuroni interni

- Gli ultimi termini sono semplicemente:

$$\frac{\partial o_j}{\partial a_j} = \frac{\partial f(a_j)}{\partial a_j} = f'(a_j) \qquad \frac{\partial a_j}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}} \sum_i w_{ji} x_i = x_i$$

- Come calcolare $\frac{\partial E}{\partial o_j}$ ?

# L'equazione di aggiornamento per i neuroni interni




Possiamo porre:

$$\frac{\partial E}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_k (d_k - o_k)^2 \quad \text{dove:} \quad o_k = f\left( \sum_j w_{kj} o_j \right)$$

# L'equazione di aggiornamento per i neuroni interni

Si ha, quindi: $\frac{\partial E}{\partial o_j} = -\sum_k (d_k - o_k) f'(a_k) w_{kj}$

Per cui, il gradiente diventa:

$$\frac{\partial E}{\partial w_{ji}} = -\left[\sum_k (d_k - o_k) f'(a_k) w_{kj}\right] f'(a_j) x_i$$

E arriviamo finalmente all'equazione di aggiornamento:

$$w_{ji}^{n+1} = w_{ji}^n + \eta \left[\sum_k (d_k - o_k) f'(a_k) w_{kj}\right] f'(a_j) x_i \Big|_{\mathbf{w}^n}$$

# Riassumendo

- Neuroni di output:

$$w_{kj}^{n+1} = w_{kj}^n + \eta (d_k - o_k) f'(a_k) x_j \Big|_{\mathbf{w}^n} = w_{kj}^n + \eta \delta_k x_j \Big|_{\mathbf{w}^n}$$

- Neuroni interni:

$$w_{ji}^{n+1} = w_{ji}^n + \eta \left[\sum_k (d_k - o_k) f'(a_k) w_{kj}\right] f'(a_j) x_i \Big|_{\mathbf{w}^n} =$$

$$w_{ji}^n + \eta \left[\sum_k \delta_k w_{kj}\right] f'(a_j) x_i \Big|_{\mathbf{w}^n} = w_{ji}^n + \eta \delta_j x_i \Big|_{\mathbf{w}^n}$$

avendo posto: $\delta_k = (d_k - o_k) f'(a_k)$ e $\delta_j = \left[\sum_k \delta_k w_{kj}\right] f'(a_j)$

# Algoritmo di Back Propagation

1. si pone in ingresso ai nodi di input un campione del TS
2. i nodi del primo livello nascosto calcolano le loro uscite che vengono poste in ingresso ai nodi del livello successivo (…)
3. i nodi di output calcolano le uscite della rete
4. conoscendo l'uscita desiderata, viene calcolato l'errore presente su ogni nodo del livello di output
5. si calcolano i $\delta_k = (d_k - o_k) f'(a_k)$ per ogni nodo del livello di output e si applica l'equazione di aggiornamento ai pesi relativi
6. per ogni nodo del penultimo livello si calcolano i $\delta_j = \left[\sum_k \delta_k w_{kj}\right] f'(a_j)$ e si applica l'equazione di aggiornamento ai pesi relativi

# Algoritmo di Back Propagation

- I passi 1-3 si indicano con il termine *forward pass*, mentre i passi 4-6 costituiscono il *backward pass*.
- L'algoritmo di apprendimento è quindi:

```
repeat
   for j=1,M
      forward pass
      backward pass
   end for
until condizione di fine learning
```

## Punti da risolvere

Alcuni punti in sospeso:

- quale funzione di uscita $f(.)$ ?
- quale condizione di fine learning ?

## La funzione di uscita

- Per quanto abbiamo visto, la funzione di uscita deve essere differenziabile
- Le funzioni maggiormente usate sono:
  - funzione lineare
  - sigmoide
  - tanh
  - gaussiana

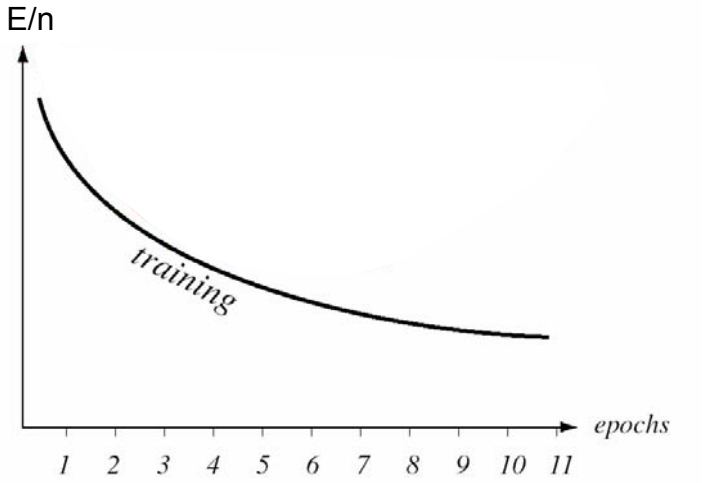
## Condizione di termine

- Valore di E
- Numero di epoche
- Andamento del gradiente
- Problemi di generalizzazione
  - validazione

## Curve di apprendimento

- All'inizio della fase di training, l'errore sul training set è tipicamente alto.
- Con il procedere dell'apprendimento, l'errore tende a diminuire, raggiungendo alla fine un valore asintotico che dipende da:
  - errore di Bayes del training set
  - dimensioni del training set
  - numero di pesi della rete
  - valore iniziale dei pesi
- L'andamento dell'errore rispetto al numero di epoche realizzate è visualizzato su una *curva di apprendimento* (*learning curve*).

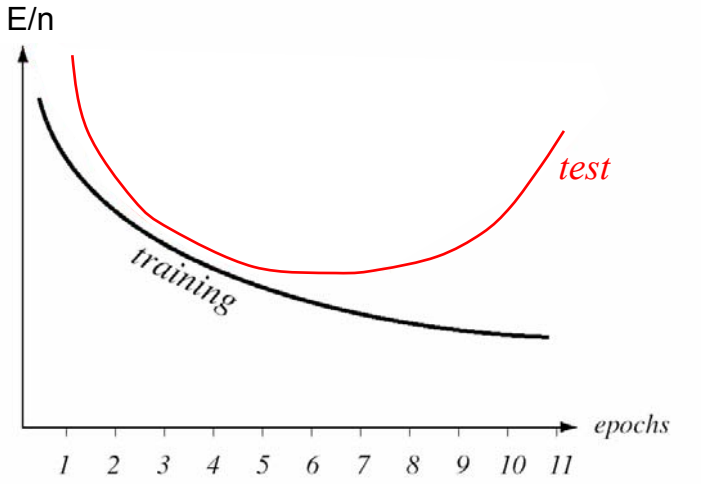
# Curve di apprendimento




# Curve di apprendimento e condizioni di termine

- Potrebbe essere possibile utilizzare le curve di apprendimento per decidere quando terminare il training ?
- A prima vista, potremmo arrestare l'apprendimento quando si è raggiunto un valore soddisfacente dell'errore o quando si raggiunge l'asintoto.
- E' davvero un buona idea ? Perché ?

# Curve di apprendimento



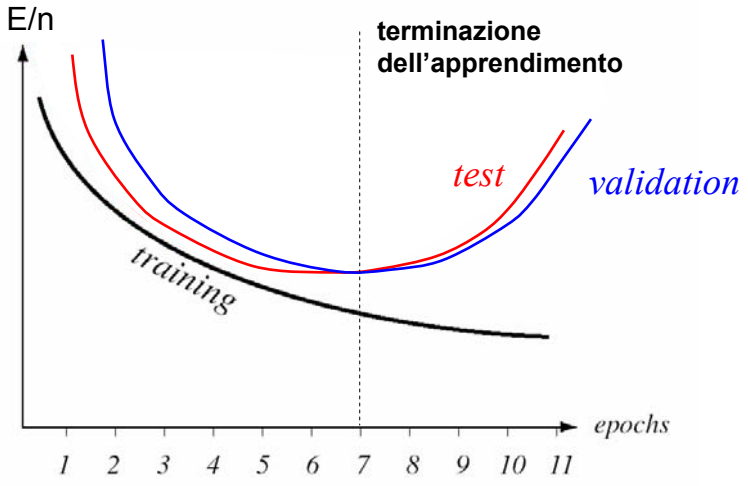

# Curve di apprendimento e condizioni di termine

- Al procedere del learning, la rete si specializza sempre meglio sul training set, migliorando l'accuratezza e garantendo una buona generalizzazione.
- Da un certo punto, però, la rete comincia a specializzarsi troppo sul training set perdendone in generalizzazione.

# Apprendimento con un insieme di validazione

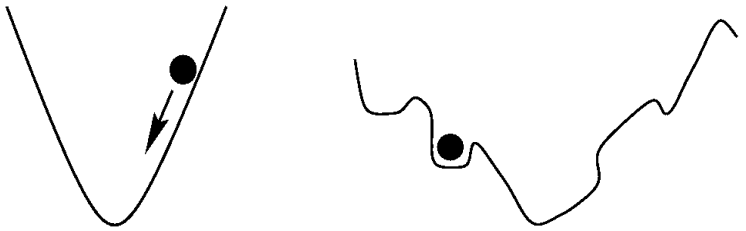
- Per controllare la capacità di generalizzazione che la rete sta acquisendo durante il training si effettua periodicamente la classificazione di un insieme di campioni non appartenenti al training set (*insieme di validazione* o *validation set*).
- L'andamento dell'apprendimento è quindi visualizzato da due curve, una valutata sul training set ed una sul validation set.
- L'arresto dell'apprendimento si può quindi realizzare in corrispondenza di un minimo sulla curva relativa al validation set.

# Terminazione dell'apprendimento



# Migliorare il training

- Problema dei minimi locali



# Migliorare il training

- Tecnica del momento

All' aggiornamento del peso, viene aggiunta una percentuale dell'aggiornamento precedente

$$\Delta w_{ji}^{n+1} = -\eta \left. \frac{\partial E}{\partial w_{ji}} \right|_{\mathbf{w}^n} + \mu \Delta w_{ji}^{n}$$

## Altri algoritmi di training

- Gradiente coniugato
- Newton
- quasi-Newton
- Levenberg-Marquardt

## Preprocessing degli ingressi

- Situazioni che possono essere dannose per il corretto apprendimento della rete:
  - Ampiezza dell'input molto elevata
  - Forte differenza tra le ampiezze dei diversi input
- Altri problemi:
  - Ingressi di tipo categorico
  - Ingressi assenti
- Preprocessing:
  - Completamento dei dati mancanti
  - Conversione categorie → valori numerici
  - normalizzazione

## Normalizzazione degli ingressi

- E' necessaria per rendere omogenei i vari ingressi ed evitare che ingressi di valore maggiore influiscano più di ingressi a valore minore.
- Una tipica normalizzazione consiste nel trasformare i valori degli ingressi in modo da rendere nulla la media e unitaria la varianza:

$$\langle x_1, x_2, \ldots, x_n \rangle \rightarrow \langle x'_1, x'_2, \ldots, x'_n \rangle$$

$$\text{dove: } x'_i = \frac{x_i - \overline{x}_i}{\sigma_{x_i}}$$

## Reti neurali e classificazione bayesiana

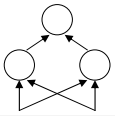
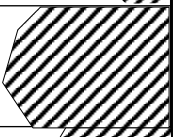
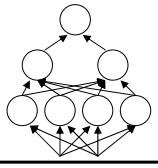
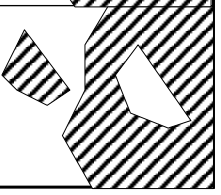
- Finora le reti neurali sono apparse come una metodologia a sé stante, indipendente dall'approccio statistico visto prima.
- In effetti, si dimostra che, in certe condizioni, l'uscita delle reti neurali approssima le probabilità a posteriori.
- Condizioni:
  - uscita 0/1;
  - apprendimento basato sul LMS;
  - dati infiniti.

# Interpretazione delle uscite come post-probabilità

- In realtà, le limitatezze del training set rendono difficile una buona approssimazione delle post-probabilità.
- Comunque, si fa uso spesso dell'uscita della rete in questo modo.
- E' necessario però assicurare che la somma delle stime sia uguale a 1.0:

$$o'_j = \frac{o_j}{\sum_j o_j} \qquad o'_j = \frac{\exp(o_j)}{\sum_j \exp(o_j)}$$

# Regioni di decisione delle reti neurali

| Struttura | Regioni di decisione | Forma generale |
|---|---|---|
|  | Semispazi delimitati da iperpiani |  |
|  | Regioni convesse |  |
|  | Regioni di forma arbitraria |  |