Reti Neurali

Apprendimento di funzioni algebriche

Storia

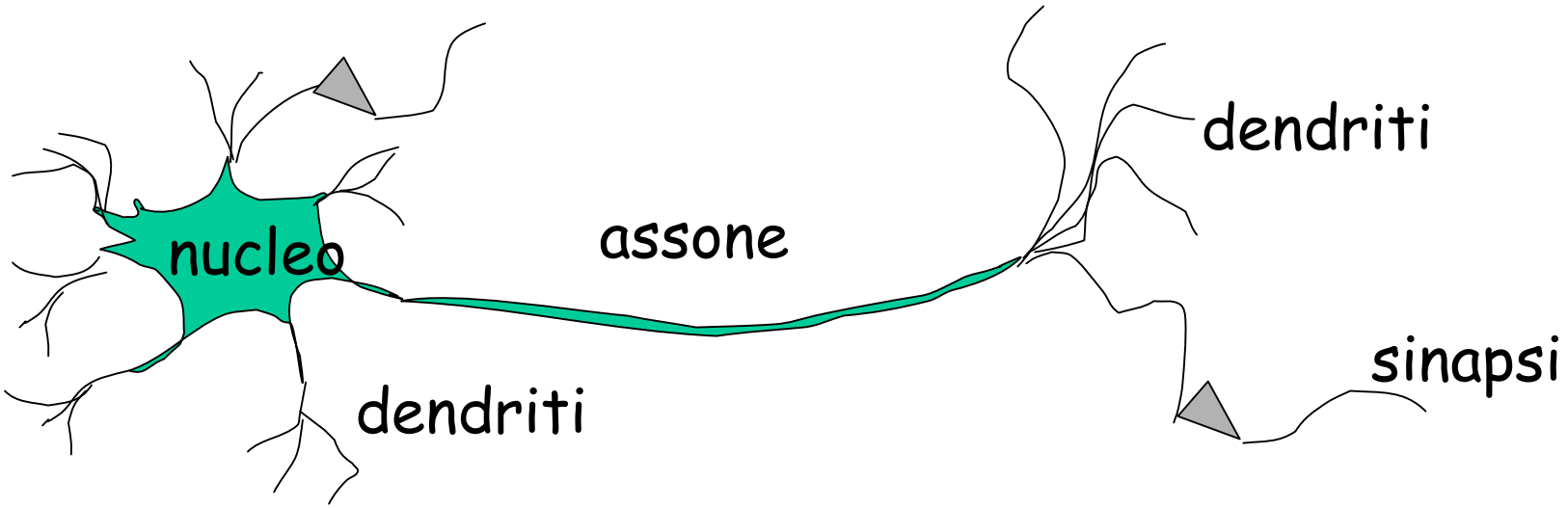
- Le reti neurali artificiali (Artificial Neural Networks o ANN) sono una simulazione astratta del nostro sistema nervoso, che contiene una collezione di neuroni i quali comunicano fra loro mediante connessioni dette assoni
- Il modello ANN ha una certa somiglianza con gli assoni e dendriti in un sistema nervoso
- Il primo modello di reti neurali fu proposto nel 1943 da McCulloch e Pitts nei termini di un modello computazionale dell'attività nervosa.
  - A questo modello sono seguiti altri proposti da John Von Neumann, Marvin Minsky, Frank Rosenblatt e molti altri

Due categorie di modelli

- **Biologico**: ha l'obiettivo di imitare sistemi neurali biologici, come le funzionalità auditive e visive.
  - Obiettivo: verifica di ipotesi riguardo ai sistemi biologici
- **Guidato dalle applicazioni**: meno interessato a "mimare" funzioni biologiche
  - Le architetture sono ampiamente condizionate dalle necessità applicative
  - Questi modelli sono anche denominati architetture connessioniste
  - Modello trattato nel corso!!!

Neuroni

- Molti neuroni possiedono strutture arboree chiamate **dendriti** che ricevono segnali da altri neuroni mediante giunzioni dette **sinapsi**
- Alcuni neuroni comunicano mediante poche sinapsi, altri ne posseggono migliaia



Funzionamento di un Neurone

- Si stima che il cervello umano contenga oltre 100 miliardi di neuroni e che un neurone può avere oltre 1000 sinapsi in ingresso e in uscita
- Tempo di commutazione di alcuni millisecondi (assai più lento di una porta logica), ma connettività centinaia di volte superiore
- Un neurone trasmette informazioni agli altri neuroni tramite il proprio assone
  - L'assone trasmette impulsi elettrici, che dipendono dal suo potenziale
  - L'informazione trasmessa può essere eccitatoria oppure inibitoria
- Un neurone riceve in ingresso segnali di varia natura, che vengono sommati
- Se l'influenza eccitatoria è predominante, il neurone si attiva e genera messaggi informativi verso le sinapsi di uscita

Struttura di una Rete Neurale

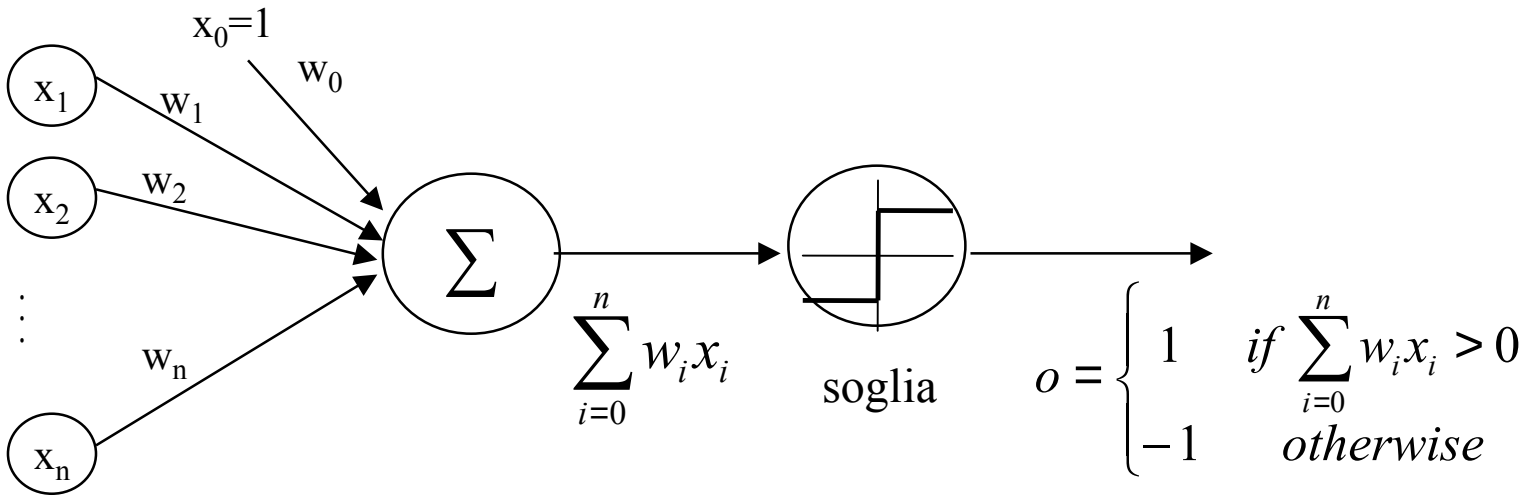
- Una **rete neurale** è costituta da:
  - un insieme di nodi (i **neuroni**) o unità connesse da collegamenti
  - Un insieme di **pesi** associati ai collegamenti
  - Un insieme di **soglie** o livelli di attivazione
- La **progettazione** di una rete neurale richiede:
  - La scelta del numero e del tipo di unità
  - La determinazione della struttura morfologica
  - Codifica degli esempi di addestramento, in termini di ingressi e uscite dalla rete
  - L'inizializzazione e l'addestramento dei pesi sulle interconnessioni, attraverso il training set

Problemi Risolvibili con le Reti Neurali

- Caratteristiche delle reti:
  - Le istanze sono rappresentate mediante **molte feature** a **molti valori**, **anche reali**
  - La funzione obiettivo può essere a **valori reali**
  - Gli esempi possono essere rumorosi
  - I tempi di addestramento possono essere lunghi
  - La valutazione della rete appresa deve poter essere effettuata velocemente
  - Non è cruciale capire la **semantica** della funzione attesa
- Applicazioni: robotica, image understanding, sistemi biologici, predizioni finanziarie, ecc.

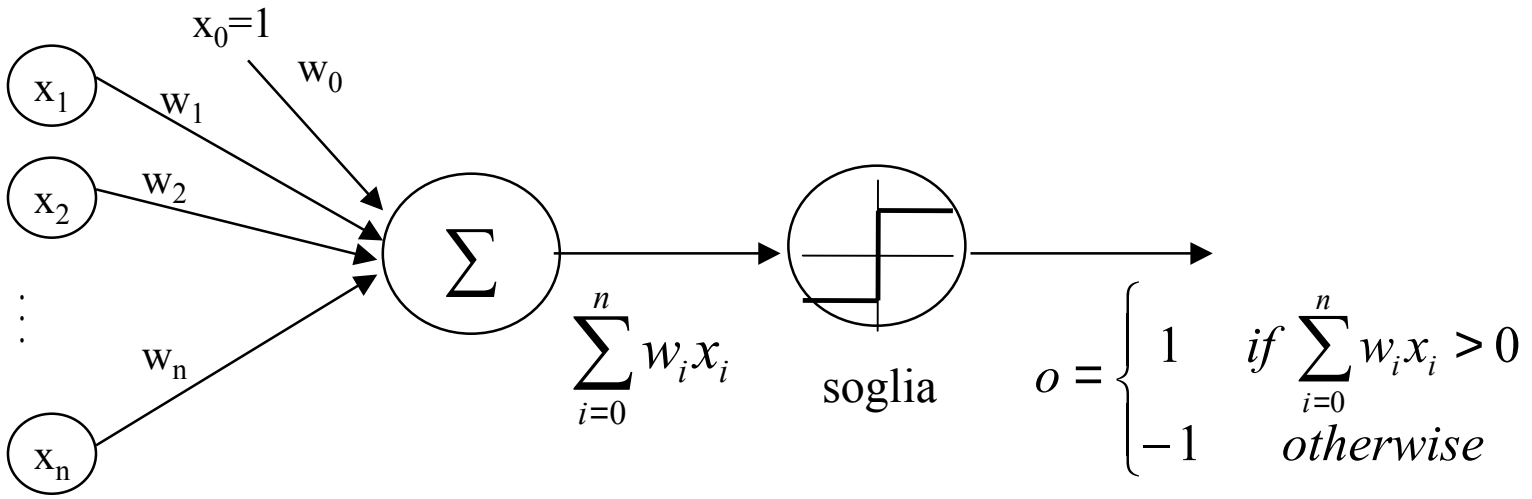
Il Percettrone

- Il percettrone è il mattone base delle reti neurali
- Nasce da un'idea di Rosenblatt (1962)
- Cerca di simulare il funzionamento del singolo neurone



Il Percettrone

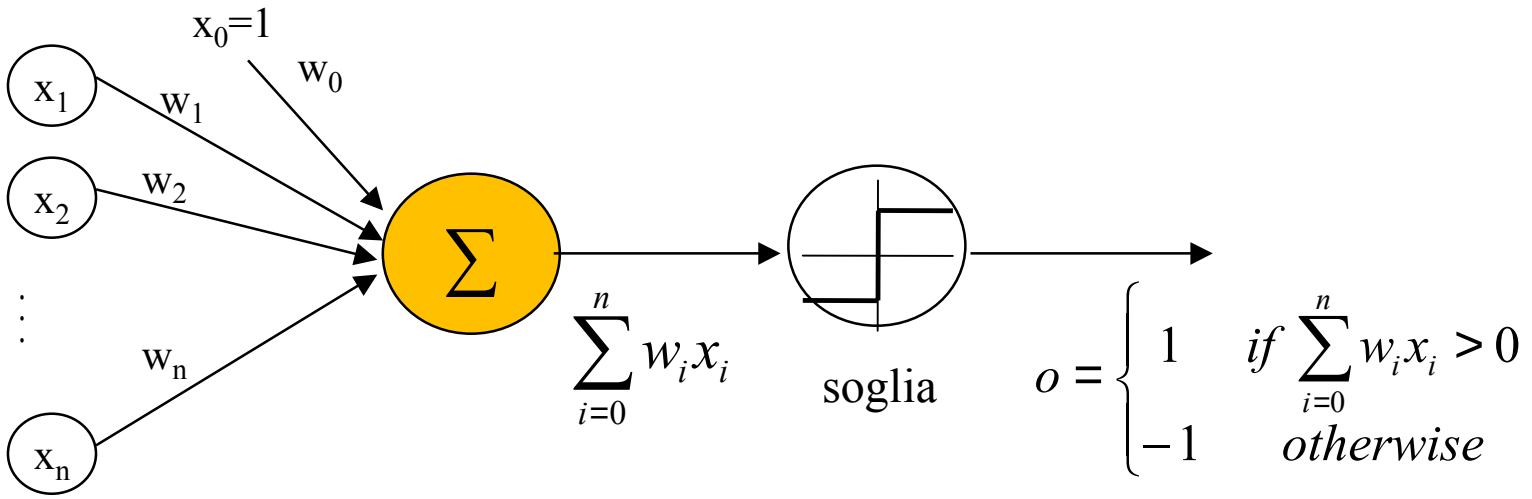
- I valori di uscita sono booleani: 0 oppure 1
- Gli ingressi x_i e i pesi w_i sono valori reali positivi o negativi
- Tre elementi: ingressi, somma, soglia
- L'apprendimento consiste nel selezionare **pesi** e **soglia**



Funzioni somma e soglia (1)

- La **funzione d'ingresso** (lineare, somma delle componenti di input di $x = (x_1, \ldots, x_n)$)

$$w_0 + w_1 x_1 + \ldots + w_n x_n = \sum_{i=0}^{n} w_i x_i = \vec{w} \cdot \vec{x}$$

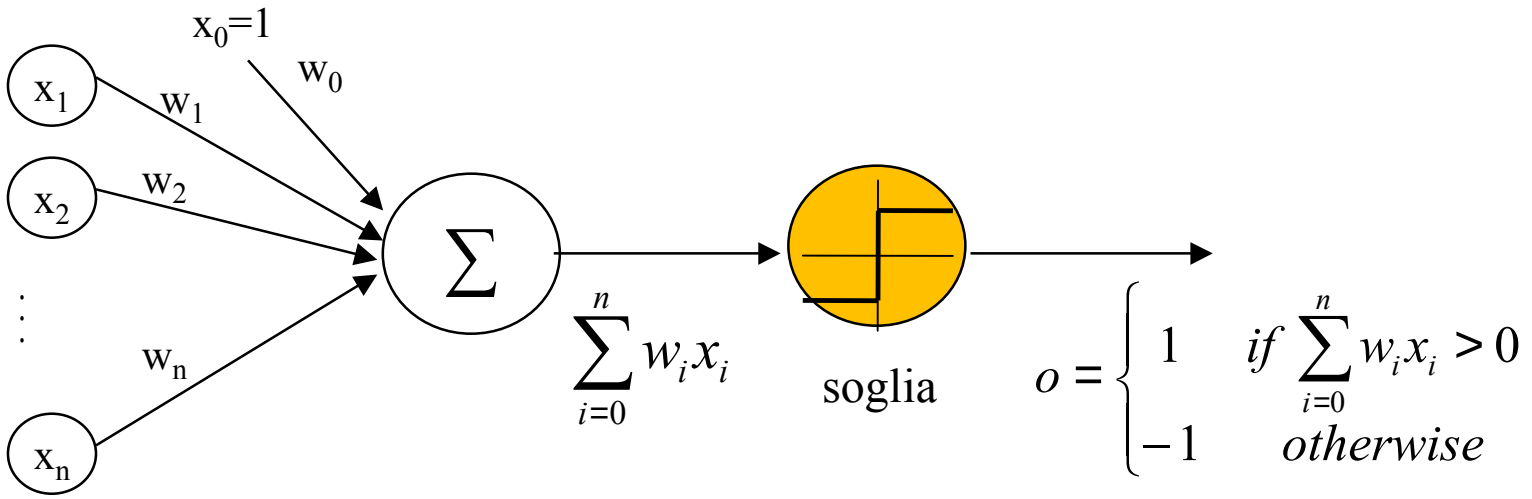


Funzioni somma e soglia (2)

- La **funzione di attivazione** (non lineare, soglia)

$$o(x_1,\ldots,x_n) = g\left(\sum_{i=0}^{n} w_i x_i\right)$$

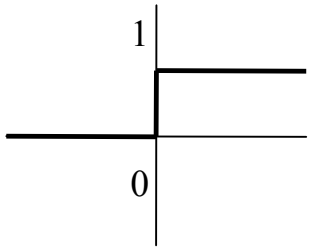
  - Vogliamo l'unità percettrone **attiva** (vicino a +1) quando gli input corretti sono forniti e **inattiva** altrimenti
  - E' preferibile che g sia **non lineare**, altrimenti la rete neurale collassa a una funzione lineare dell'input



Funzioni di attivazione

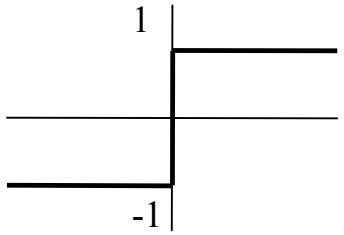
- Funzione **gradino**

$$gradino_t(x) = \begin{cases} 1 & x > t \\ 0 & altrimenti \end{cases}$$



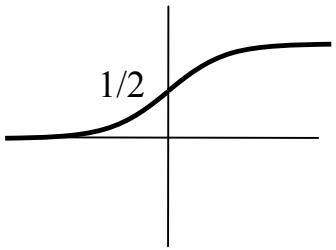
- Funzione **segno**

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



- Funzione **sigmoide**

$$sigmoide(x) = \frac{1}{1 + e^{-x}}$$



Funzione obiettivo

- Ad esempio, se la funzione soglia è la funzione segno e $x_1, ..., x_n$ sono i valori degli attributi delle istanze x di X, la funzione obiettivo da apprendere è:, si ha:

$$o(\vec{x}) = \begin{cases} 1 & se\ w_0x_0 + w_1x_1 + ... + w_nx_n > 0 \\ -1 & altrimenti \end{cases}$$

- Esprimibile anche in forma vettoriale mediante la:

$$o(\vec{x}) = sign(\vec{w} \cdot \vec{x})$$

Il Percettrone

- Problema di apprendimento:
  - dati insiemi di punti su uno spazio n-dimensionale, classificarli in due gruppi (positivi e negativi)
  - inoltre dato un nuovo punto x’ decidere a quale gruppo appartiene
- Il primo problema è di **classificazione**, mentre per risolvere il secondo è richiesta capacità di **generalizzazione**, come all'apprendimento di concetti

Problema di classificazione

- Il problema è quindi ridotto alla determinazione dell'insieme dei pesi (w_0, w_1, …, w_n) migliore per minimizzare gli errori di classificazione
- Quindi lo spazio delle ipotesi H è infinito ed è dato da tutte le possibili assegnazioni di valori agli n+1 pesi (w_0, w_1, …, w_n):

$$H = \{\vec{w} : \vec{w} \in \Re^{n+1}\}$$

- Si tratta di ottimizzare la funzione

$$o(\vec{x}) = sign(\vec{w} \cdot \vec{x})$$

Esempio per due attributi

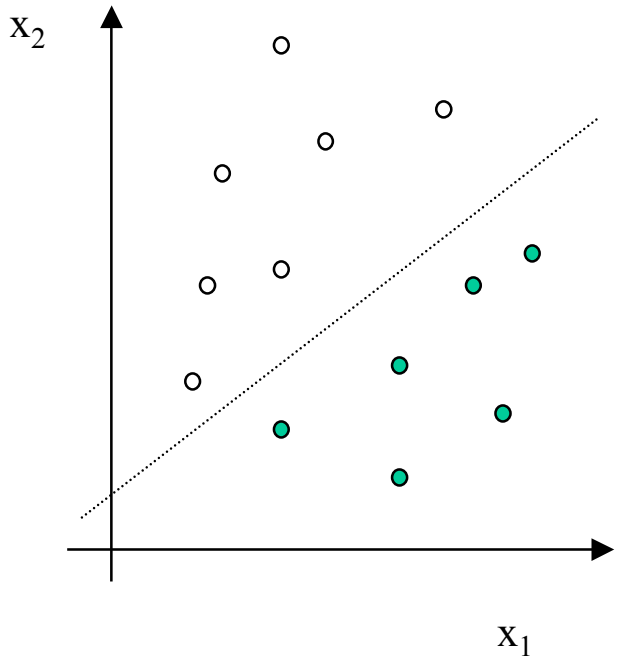
- Con (x_1, x_2) in ingresso, si ha:

$$o(\vec{x}) = sign(w_0 + w_1 x_1 + w_2 x_2)$$

- La retta di separazione è data da:

$$w_0 + w_1 x_1 + w_2 x_2 = 0$$

$$x_2 = -\frac{w_1}{w_2} x_1 - \frac{w_0}{w_2}$$



- Nel caso con n attributi, quel che si apprende è un **iperpiano di separazione** dato da:

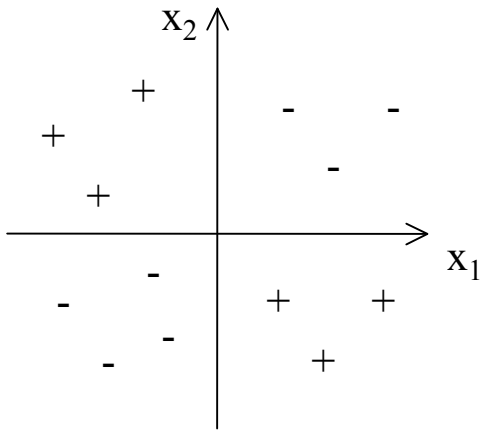
$$\vec{w} \cdot \vec{x} = 0$$

Funzioni rappresentabili con il percettrone

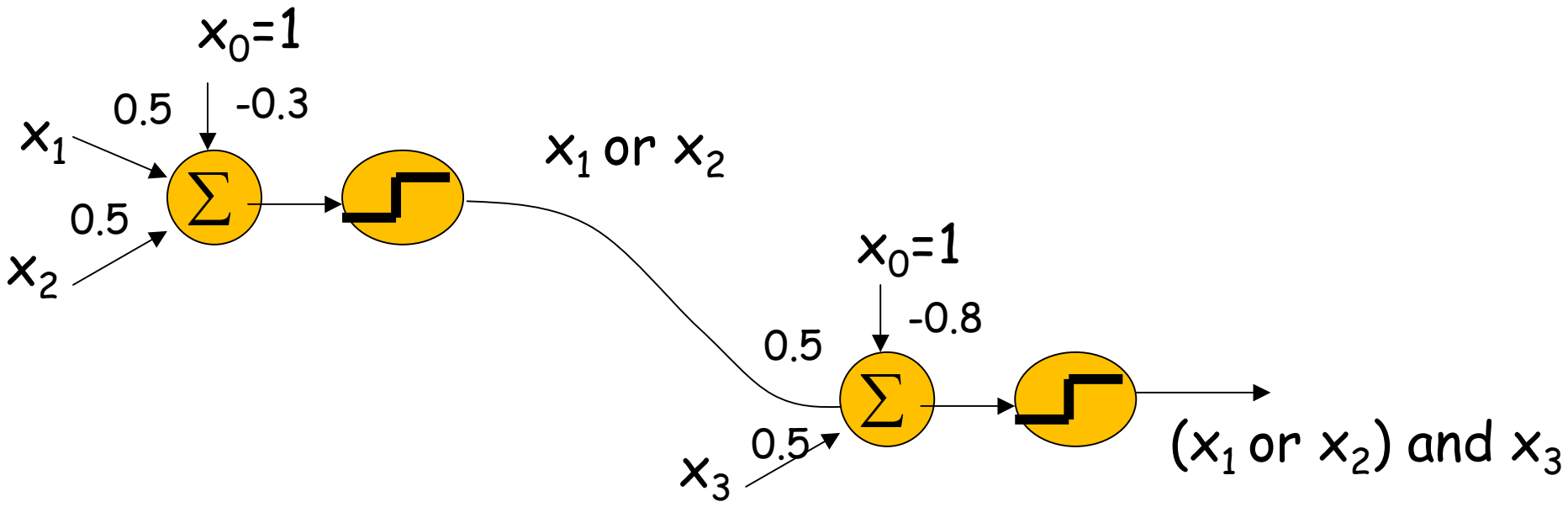
- I percettroni possono rappresentare tutte le funzioni booleane primitive AND, OR, NAND e NOR
  - $AND(x_1, x_2) = sign(-0.8 + 0.5x_1 + 0.5x_2)$
  - $OR(x_1, x_2) = sign(-0.3 + 0.5x_1 + 0.5x_2)$
  - ecc. (provate a scrivere le altre per esercizio)

Funzioni rappresentabili con il percettrone

- I percettroni possono rappresentare tutte le funzioni booleane primitive AND, OR, NAND e NOR
- Alcune funzioni booleane non possono essere rappresentate
  - es. la funzione XOR (vale 1 se e solo se $x_1 \neq x_2$):



Implementare funzioni booleane più complesse con il percettrone



Algoritmo di Addestramento del Percettrone

- Inizializza i pesi casualmente
- Sottoponi un esempio (x, t(x)) $\in$ D, ovvero t(x) è il l'output effettivo
- Calcola la funzione o(x)
- Se o(x) $\neq$ t(x) allora aggiorna il peso dell'i-esimo attributo:

$$w_i = w_i + \Delta w_i$$

$$\Delta w_i = \eta(t(\vec{x}) - o(\vec{x}))x_i$$

  - η si chiama *learning rate*
  - x_i è il valore dell'attributo i-esimo di x
  - la quantità t-o rappresenta l'errore E del percettrone

Esempio

- Supponiamo o(x)=-1 (se la funzione soglia è sign(x)) e t(x)=1
- Bisogna modificare i pesi per accrescere il valore di $\vec{w} \cdot \vec{x}$
- Se per esempio:

$$x_i = 0{,}8, \eta = 0{,}1, t = 1, o = -1$$

$$\Delta w_i = \eta(t - o)x_i = 0{,}1(1 - (-1))0{,}8 = +0{,}16$$

- Quindi il valore dei w_i tenderà a crescere, riducendo l'errore
- Se invece t(x)=-1 e o(x)=1

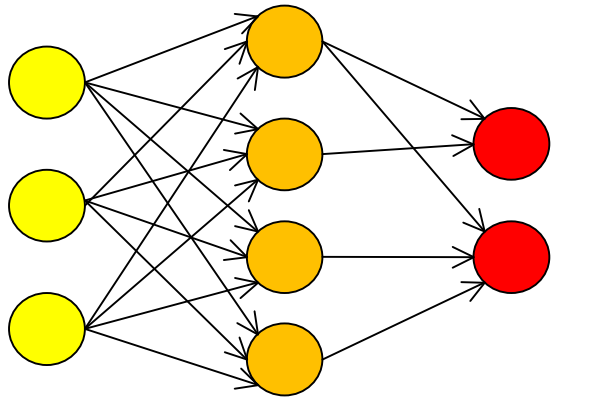
$$\Delta w_i = \eta(t - o)x_i = 0{,}1(-1 - (+1))0{,}8 = -0{,}16$$

Convergenza del Percettrone

- Il **teorema di convergenza del percettrone** (Rosemblatt, 1962) assicura che il percettrone riuscirà a delimitare le 2 classi se il sistema è ***linearmente separabile***
- In altre parole, nell'ottimizzazione non esistono minimi locali
- Problema: come ci si comporta in caso di situazioni **non linearmente separabili**?
- Soluzioni: **reti multistrato alimentate in avanti, e reti ricorrenti**

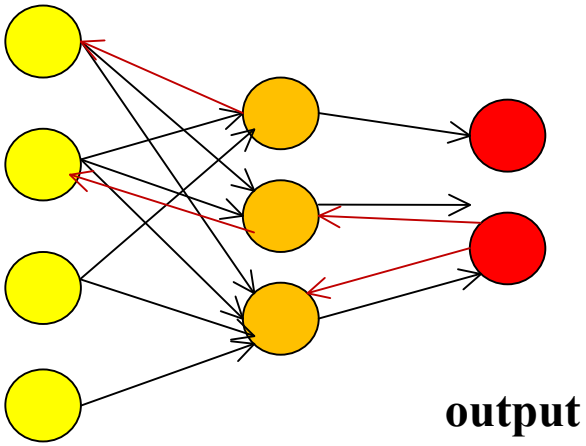
Soluzione 1:
Reti Stratificate Alimentate in Avanti

- Feedforward Neural Networks
  - Ogni unità è collegata solo a quella dello strato successivo
  - L'elaborazione procede uniformemente dalle unità di ingresso a quelle di uscita
  - Non c'è feedback (grafo aciclico diretto o DAG)
  - Non hanno stato interno



Soluzione 2: Reti Ricorrenti

- Sono più simili al funzionamento del cervello umano, in quanto simulano le connessioni all'indietro
  - Nel cervello esistono evidenti connessioni all'indietro
- I collegamenti possono formare configurazioni arbitrarie
- L'attivazione viene "ripassata" indietro alle unità che l'hanno provocata
- Hanno uno stato interno: i livelli di attivazione
- Computazione meno ordinata
- Instabili
- Più tempo per calcolare lo stato stabile
- Difficoltà nell'apprendimento
- Implementano agenti più complessi
- Esempi:
  - Reti di Boltzmann
  - Reti di Hopfield



Reti Alimentate in Avanti: Algoritmo di Backpropagation

- Apprendimento con **algoritmo di backpropagation** (propagazione all'indietro)
- Obiettivi:
  - partire da un insieme di percettroni con pesi casuali
  - apprendere in modo veloce
  - avere capacità di generalizzazione
  - avere insiemi di percettroni su larga scala

Backpropagation (2)

- La funzione soglia di ciascuna unità è la sigmoide:

$$o(\vec{x}) = \sigma(\vec{w} \cdot \vec{x}) = \frac{1}{1 + e^{-\vec{w} \cdot \vec{x}}}$$

- L'errore è calcolato come segue:

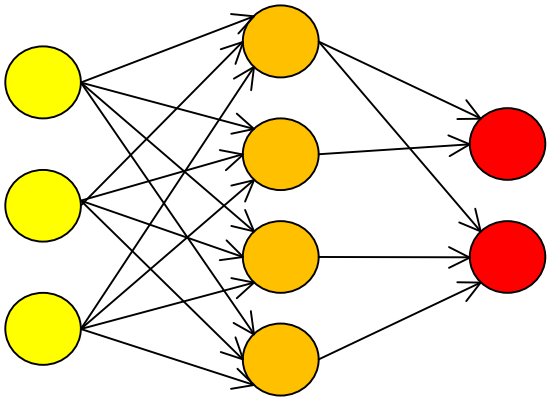
$$E(\vec{w}) = \frac{1}{2} \sum_{(\vec{x},\vec{t}) \in D} \sum_{k \in outputs} (t_k - o_k)^2$$

- **Obiettivo** dell'algoritmo: minimizzare l'errore fra ciascuna uscita desiderata t_k e l'uscita calcolata dalla rete neurale o_k

unità d'ingresso i_i

unità nascosta h_j

unità di uscita o_k



L'algoritmo Backpropagation

function BackProp(D, η, n_{in}, n_{hidden}, n_{out})

- D è l'insieme di addestramento costituito da coppie $(\vec{x}, \vec{t})$
- η è il tasso di apprendimento (learning rate), es. 0.1
- n_{in}, n_{hidden} e n_{out} sono il numero di unità d'ingresso, nascoste e d'uscita

- Crea una rete feed-forward con n_{in}, n_{hidden} e n_{out} unità
- Inizializza tutti i pesi con numeri casuali piccoli (es. tra -0.05 e 0.05)
- Finché non si incontra la condizione di terminazione:
- Per ogni esempio $(\vec{x}, \vec{t})$ in D:
  - Propaga l'input in avanti sulla rete calcolando l'output o_u di ogni unità u della rete
  - Propaga gli errori all'indietro nella rete:
    - Per ogni unità di output k, calcola il suo termine d'errore δ_k:

$$\delta_k = o_k(1 - o_k)(t_k - o_k)$$

    - Per ogni unità nascosta h, calcola il suo termine d'errore δ_h:

$$\delta_h = o_h(1 - o_h) \sum_{k \in outputs} w_{kh}\delta_k$$

    - Aggiorna i pesi w_{ji}:

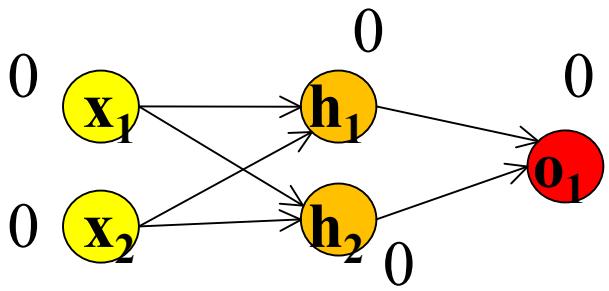
$$w_{ji} = w_{ji} + \Delta w_{ji}, \qquad \text{dove } \Delta w_{ji} = \eta \delta_j x_{ji}$$

  - x_{ji} è l'input dell'unità j proveniente dall'unità i

Esempio

- Crea una rete feed-forward con n_{in}, n_{hidden} e n_{out} unità
- Inizializza tutti i pesi con numeri casuali piccoli (es. tra -0.05 e 0.05)

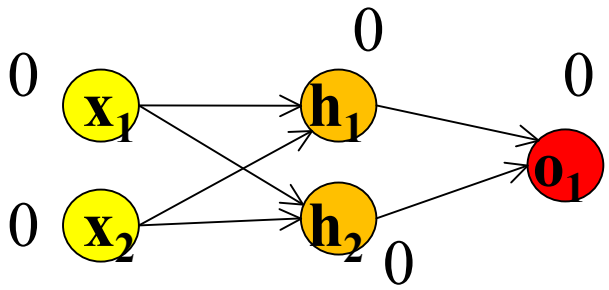
- $w_{h1x0} = 0.34$, $w_{h1x1} = 0.12$, $w_{h1x2} = -0.92$
- $w_{h2x0} = -0.11$, $w_{h2x1} = 0.57$, $w_{h2x2} = -0.32$
- $w_{o1h0} = -0.99$, $w_{o1h1} = 0.16$, $w_{o1h2} = 0.75$



Esempio

- Finché non si incontra la condizione di terminazione:
  - Per ogni esempio $(\vec{x}, \vec{t})$ in D: (x=(0, 0), t=0)
    - Propaga l'input in avanti sulla rete calcolando l'output o_u di ogni unità u della rete

    - $o_{h1} = \sigma(0.34*\mathbf{1} + 0.12*0 + -0.92*0) = \sigma(0.34) = 1/(1+e^{-0.34}) = 0.58$
    - $o_{h2} = \sigma(-0.11*\mathbf{1} + 0.57*0 + -0.32*0) = \sigma(-0.11) = 1/(1+e^{-(-0.11)}) = 0.47$
    - $o_{o1} = \sigma(-0.99*\mathbf{1} + 0.16*0.58 + 0.75*0.47) = \sigma(-0.54) = 1/(1+e^{-0.54}) = 0.36$



Esempio

- Propaga gli errori all'indietro nella rete:
  - **Per ogni unità di output k, calcola il suo termine d'errore δ_k:**

$$\delta_k = o_k(1-o_k)(t_k - o_k)$$

  - **Per ogni unità nascosta h, calcola il suo termine d'errore δ_h:**

$$\delta_h = o_h(1-o_h)\sum_{k \in outputs} w_{kh}\delta_k$$

  - Aggiorna i pesi w_{ji}:

$$w_{ji} = w_{ji} + \Delta w_{ji}, \qquad \text{dove } \Delta w_{ji} = \eta \delta_j x_{ji}$$

- $\delta_{o1} = o_{o1}(1\text{-}o_{o1})(t_{o1}\text{-}o_{o1}) = 0.36*(1\text{-}0.36)(0\text{-}0.36) = \text{-}0.08$
- $\delta_{h1} = o_{h1}*(1\text{-}o_{h1})*w_{o1h1}\delta_{o1} = 0.58*(1\text{-}0.58)*(0.16)*(\text{-}0.08) = \text{-}0.003$
- $\delta_{h2} = o_{h2}*(1\text{-}o_{h2})*w_{o1h2}\delta_{o1} = 0.47(1\text{-}0.47)*(0.75)*(\text{-}0.08) = \text{-}0.014$

Esempio

- Propaga gli errori all'indietro nella rete:
  - Per ogni unità di output k, calcola il suo termine d'errore δ_k:

$$\delta_k = o_k(1-o_k)(t_k - o_k)$$

  - Per ogni unità nascosta h, calcola il suo termine d'errore δ_h:

$$\delta_h = o_h(1-o_h)\sum_{k \in outputs} w_{kh}\delta_k$$

  - **Aggiorna i pesi w_{ji}:**

$$w_{ji} = w_{ji} + \Delta w_{ji}, \qquad \text{dove } \Delta w_{ji} = \eta \delta_j x_{ji}$$

- $w_{h1x0} = 0.34+\eta\delta_{h1}x_{h1x0}=0.34+0.5*(-0.003)*1=0.34-0.0015$,
- $w_{h1x1} = 0.12+0$, $w_{h1x2} = -0.92+0$
- $w_{h2x0} = -0.11+\eta\delta_{h2}x_{h2x0}=0.34+0.5*(-0.014)*1=0.34-0.007$,
- $w_{h2x1} = 0.57+0$, $w_{h2x2} = -0.32+0$
- $w_{o1h0} = -0.99+\eta\delta_{o1}x_{o1h0}=-0.99+0.5*(-0.08)*1=-0.99-0.04$
- $w_{o1h1} = 0.16+\eta\delta_{o1}x_{o1h1}=0.16+0.5*(-0.08)*0.58=0.16-0.02$
- $w_{o1h2} = 0.75+\eta\delta_{o1}x_{o1h2}=0.75+0.5*(-0.08)*0.47=0.75-0.02$

Esempio

- Si aggiornano tutti pesi
- Si considera il secondo esempio di D
- Si ricalcolano input e output di tutti i nodi
- Si ricalcolano gli errori sui nodi
- Si riaggiornano i pesi
- Finché non si esaurisce D (Epoca)
- Si ripete per n epoche, fino a che l'errore non si stabilizza

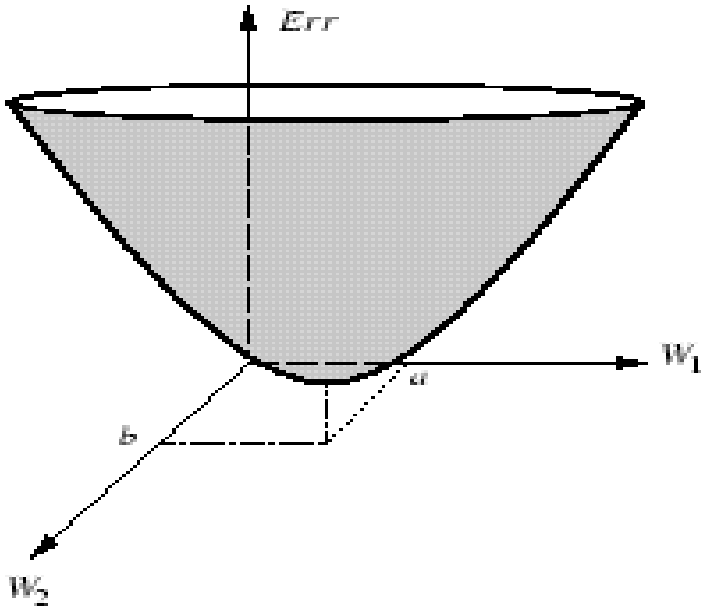
Esempio

- Dopo qualche migliaio di iterazioni su tutti e quattro gli esempio di D = { ((0,0), (0)), ((0,1), (1)), ((1,0), (1)), ((1,1),(0))}
- Otteniamo i seguenti risultati per gli input:
  - x = (0, 0), o_{o1} = 0.017622
  - x = (0, 1), o_{o1} = 0.981504
  - x = (1, 0), o_{o1} = 0.981491
  - x = (1, 1), o_{o1} = 0.022782

La rete dell'esempio calcola la funzione XOR!!!

http://delfin.unideb.hu/~bl0021/xordemo/xordemo.html
http://www.generation5.org/content/2001/xornet.asp

Spiegazione della regola di propagazione dell'errore all'indietro



Consideriamo la funzione di errore per $x \in D$:

$$E_x = \frac{1}{2} \sum_i (t_i - o_i)^2$$

Supponiamo di dover imparare solo due pesi.

Il piano w_1 w_2 rappresenta lo spazio delle ipotesi (pesi delle connessioni) , il cono è la superficie d'errore.

Per minimizzare l'errore si deve calcolare la direzione della discesa più ripida lungo la superficie.

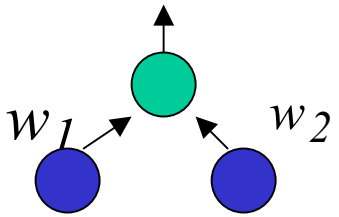
Quindi, la derivata.

Esempio di calcolo del gradiente

$$\Delta w_1 = -\eta \frac{\partial E(w_1 x_1 + w_2 x_2)}{\partial w_1} = -\eta \frac{\partial E}{\partial net_1} \frac{\partial net_1}{\partial w_1} = -\eta \frac{\partial E}{\partial net_1} x_1$$

$$net_1 = w_1 x_1 + w_2 x_2$$

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



$$\frac{\partial E}{\partial net_1} = \frac{\partial E}{\partial o} \frac{\partial o}{\partial net_1} \qquad \frac{\partial E}{\partial o} = \frac{1}{2} 2(t - o) \frac{\partial (t - o)}{\partial o} = -(t - o)$$

$$\frac{\partial o}{\partial net_1} = \frac{\partial \sigma(net_1)}{\partial net_1} = o(1 - o)$$

$$\partial(\sigma(x)) = \sigma(x)(1 - \sigma(x))$$

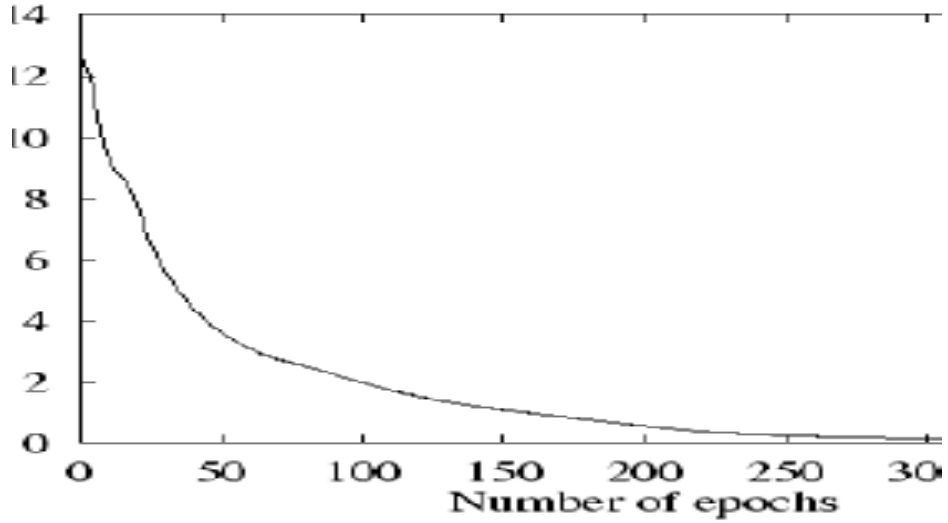
$$\Delta w_1 = \eta o(1 - o)(t - o) x_1$$

È la formula usata dall'algoritmo BP!!

Condizioni di terminazione

- Il processo continua finchè non sono esauriti tutti gli esempi (*epoca*), poi si riparte
- Quando arrestare il processo? Minimizzare gli errori sul set D non è un buon criterio (*overfitting*)
- Si preferisce minimizzare gli errori su un test set T, cioè suddividere D in D'$\cup$T, addestrare su D' e usare T per determinare la condizione di terminazione.

Plot dell'errore su un test set T



Ma l'algoritmo converge?

- Problemi dell'algoritmo del gradiente:
  - Può arrestarsi su minimi locali
  - Un minimo locale può dare soluzioni molto peggiori del minimo globale
  - Possono esserci molti minimi locali
- Soluzioni possibili: addestra la rete con pesi iniziali diversi, addestra diverse architetture di rete

Modifica della regola di aggiornamento

- Quando viene osservato l'esempio n-esimo di D, ~~la regola~~ di aggiornamento diventa: $\Delta wij(n) \leftarrow \eta\delta_j x_{ij} + \alpha\Delta w_{ij}(n-1)$
- Il secondo termine prende il nome di *momento*.
- Vantaggi:
  - È possibile superare minimi locali
  - Mantiene i pesi nelle zone dove l'errore è piatto
  - Aumenta la velocità dove il gradiente non cambia
- Svantaggi:
  - Se il momento è troppo alto, si può cadere in massimi locali
  - E' un parametro in più da regolare!

Alcune considerazioni pratiche

- La scelta dei pesi iniziali ha un grande impatto sul problema della convergenza! Se la dimensione dei vettori di input è N ed N è grande, una buona euristica è scegliere i pesi iniziali fra -1/N e 1/N
- L'algoritmo BP è molto sensibile al fattore di apprendimento η. Se è troppo grande, la rete diverge.
- A volte, è preferibile usare diversi valori di η per i diversi strati della rete
- La scelta della modalità di codifica degli ingressi e della architettura della rete può influenzare in modo drastico le prestazioni!!

Conclusioni

- Struttura ottimale della rete:

Come tutti i modelli statistici, anche le reti neurali sono soggette a sovradattamento.
In questo caso, il sovradattamento può essere causato da una struttura di rete non efficiente, troppo "piccola" o troppo "grande".
Non esiste alcuna buona teoria per caratterizzare le funzioni rappresentabili efficientemente tramite reti!
(una possibile risposta è rappresentata dagli **algoritmi genetici**)

- Espressività:

Non hanno il potere espressivo delle rappresentazioni logiche (come gli alberi di decisione). Ma a differenza di questi ultimi, sono adatte a rappresentare funzioni per ingressi e uscite di tipo continuo. In termini molto generici, sono adatte per funzioni per le quali le interazioni fra ingressi non sono "intricate" e l'uscita varia gradualmente al variare degli ingressi. Inoltre, tollerano bene il rumore.

- Efficienza computazionale:

Per m esempi e $|W|$ pesi, ogni epoca richiede un tempo $O(m|W|)$.

- Trasparenza: le reti neurali sono "scatole nere". Non hanno semantica!

Un esempio

Riconoscimento di fisionomie (face recognition)

http://www2.cs.cmu.edu/afs/cs.cmu.edu/user/avrim/www/ML94/face_homework.html

Il compito:

Classificare immagini video di visi di persone in varie pose.

Data Set:

Immagini di 20 persone in (circa) 32 pose, variabili per:
espressione,
direzione dello sguardo, occhiali da sole (si/no), sfondo

Funzioni "target""Riconoscere" chi porta occhiali, in che direzione guarda, quale persona é..

Scelte di progetto

Accuratezza della codifica di ingresso:

Una scelta chiave consiste nel decidere quali e quante "features" codificare. Ad esempio, si potrebbe pre-analizzare l'immagine estraendo contorni, regioni di intensità luminosa uniforme, ecc.

Problema: il numero delle features sarebbe variabile (ad esempio, i contorni) mentre una NN ha un numero fisso di unità di ingresso.
➔ codifica l'immagine come un insieme di valori di intensità di pixels 30x32 (un input per pixel)

Codifica di uscita:
Dipende dalla funzione. Ad esempio, vogliamo sapere se la persona guarda a sinistra, destra, dritto, o in alto.
Possiamo usare 4 uscite (1-out-of-n output encoding) oppure definire una sola uscita con 4 ranges di valori.
La prima soluzione dà più gradi di libertà alla rete (ci sono più "pesi" da determinare). Inoltre, la differenza fra i valori degli output è una misura della "confidenza".

Struttura della rete

- L'algoritmo di Backpropagation può essere applicato a qualsiasi grafo aciclico diretto, con funzione di attivazione sigmoidale.
- La struttura più comune è stratificata, con un numero di strati interni da uno a tre (>3 i tempi di training diventano inaccettabili), in cui ogni unità dello strato i è collegata con ogni unità dello strato i+1.

Quanti strati interni, quante unità in ogni strato?

Esperimento: con solo 3 unità nascoste e un solo strato interno, 90% precisione. Con 30 unità, solo +2%
Con 260 immagini per il training, 1 ora di tempo di addestramento su una Sun Sparc (30 unità) e solo 5 minuti con 3 unità.

Questo è un risultato abbastanza generale: un numero piccolo di unità nascoste consente di ottenere buoni risultati. Piccoli miglioramenti si ottengono a fronte di grosse variazioni dei parametri, causando, se non si prendono gli accorgimenti necessari, tendenza a overfitting.

Altri parametri di apprendimento

- η (tasso di apprendimento)=0,3
- α (momentum) = 0,3
- I pesi iniziali sulle unità di uscita sono *random*
- Le unità di ingresso sono inizializzate con un peso 0, perché ciò porta ad una migliore visualizzazione dei pesi appresi.

Overfitting, generalizzazione, criteri di stop

Un problema importante è capire quando arrestare il processo di terazione (stopping criterion).

Osservare solo l'andamento dell'errore è una strategia non buona.
Al crescere delle iterazioni, il sistema cerca di adattarsi anche a esempi idiosincratici, generando una ipotesi molto complessa. Accade frequentemente che questa ipotesi perda di generalità, ovvero, commetta parecchi errori nel predire casi non visti.
Osservate come, sul test set, la probabilità di errore cresca, benché diminuisca sul training set.
Un buon metodo è la cross-validation.
I pesi con i valori migliori vengono usati per calcolare le prestazioni su un set di validazione, diverso da quello di apprendimento.
Il traing termina quando i risultati cominciano a divergere.

Dopo alcune iterazioni...

Immagini apprese dopo 100 iterazioni