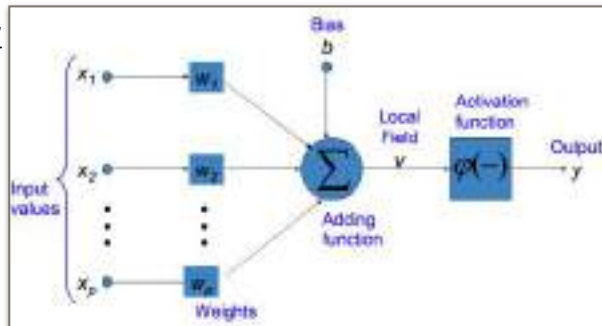


TIPI DI RETI NEURALI

1. PERCETTRONE

Struttura:



Tipo: FEED-FORWARD

Archi: MONODIREZIONALI

Bias: sì, 1 per il neurone

Livelli: INPUT + OUTPUT

F. attivazione:

$$\varphi(v) = \begin{cases} a & \text{if } v < c \\ b & \text{if } v > c \end{cases}$$

$c=0$, $a=1$, $b=-1$ v =somma pesata (input*pesi+bias)

Apprendimento/Correzione dell'errore/Formule:

$$w(n+1) = w(n) + \eta d(n)x(n);$$

SUPERVISIONATO

η = learning rate , $d(n)$ = valore atteso , $x(n)$ = valore ottenuto

Terminazione dell'apprendimento: nessun esempio è più mal-classificato / viene raggiunto il numero fissato di step-epoche / teorema di convergenza

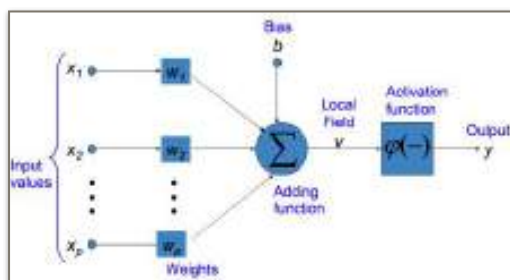
Peculiarità: —

Difetti: non risolve problemi non linearmente separabili (es: XOR)

Utilizzi: problemi linearmente separabili (es: AND)

2. ADALINE

Struttura:



(ma con correzione dell'errore: delta rule)

Tipo: FEED-FORWARD

Archi: MONODIREZIONALI

Bias: si, 1 per il neurone

Livelli: INPUT+OUTPUT

F. attivazione:

$$v = \sum_{i=0}^p w_i x_i$$

v=netinput della rete = somma pesata

Apprendimento/Correzione dell'errore/Formule: SUPERVISIONATO

For the k-th example: $(x^{(k)}, d^{(k)})$

$$E^{(k)}(w) = \frac{1}{2} (d^{(k)} - y^{(k)})^2$$

Total error:

$$E_{\text{tot}} = \sum_{k=1}^N E^{(k)}$$

(CALCOLO DELL'ERRORE)

$$\frac{\partial E^{(k)}(w)}{\partial w_j} = \frac{\partial E^{(k)}(w)}{\partial y^{(k)}} \frac{\partial y^{(k)}}{\partial w_j} = -(d^{(k)} - y^{(k)}) x_j^k$$

• Delta rule for weight update:

$$w(k+1) = w(k) + \eta (d^{(k)} - y^{(k)}) x^{(k)}$$

(WEIGHT UPDATE)

Terminazione dell'apprendimento: errore scende sotto una soglia prefissata / viene raggiunto il numero fissato di step-epoche

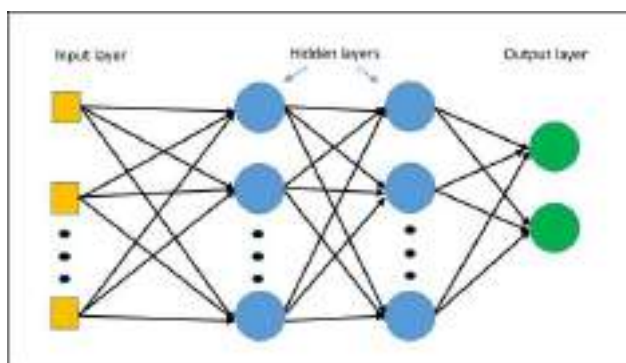
Peculiarità: —

Difetti: non risolve problemi non linearmente separabili (es: XOR)

Utilizzi: problemi linearmente separabili (es: AND)

3. MULTI-LAYER PERCEPTRON (MLP)

Struttura:



Tipo: FEED-FORWARD Archi: MONODIREZIONALI

Bias: sì, 1 per ogni neurone

Livelli: INPUT+ n HIDDEN (n max= 2) + OUTPUT

F. attivazione:

$$\varphi(v_j) = \frac{1}{1 + e^{-av_j}} \text{ with } a > 0$$

$$\text{where } v_j = \sum_i w_{ji} y_i$$

with w_{ji} weight of link from node i to node j and y_i output of node i

== input per il neurone j

Apprendimento/Correzione dell'errore/Formule:

- The error of output neuron j after the activation of the network on the n -th training example $(x(n), d(n))$ is:

$$e_j(n) = d_j(n) - y_j(n)$$

- The pattern error is the sum of the squared errors of the output neurons:

$$E(n) = \frac{1}{2} \sum_{\text{output node } j} e_j^2(n)$$

- The total mean squared error is the average of the network errors of the training examples.

$$E_{AV} = \frac{1}{N} \sum_{n=1}^N E(n)$$

$$w_{ji} = w_{ji} + \Delta w_{ji}$$

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}} \quad \eta > 0$$

$$\delta_j = -\frac{\partial E}{\partial v_j}$$

Delta rule $\Delta w_{ji} = \eta \delta_j y_i$ i = neurone dello strato prima di j

$$\delta_j = \begin{cases} \varphi'(v_j)(d_j - y_j) & \text{IF } j \text{ output node} \\ \varphi'(v_j) \sum_k \delta_k w_{kj} & \text{IF } j \text{ hidden node} \end{cases}$$

$$\text{where } \varphi'(v_j) = ay_j(1 - y_j)$$

$$N > \frac{|W|}{(1 - a)} \quad \begin{array}{l} |W| = \text{number of weights} \\ a = \text{expected accuracy on test set} \end{array}$$

SUPERVISIONATO

Terminazione dell'apprendimento: total mean squared error scende sotto una soglia prefissata / viene raggiunto il numero fissato di step-epoche

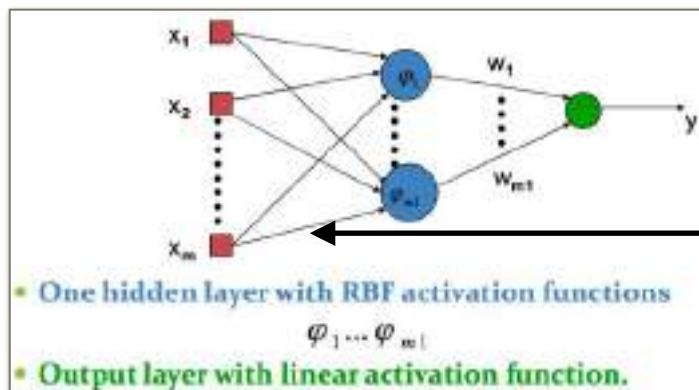
Peculiarità: teorema di approssimazione universale: a single hidden layer is sufficient for a MLP to compute a uniform approximation to a given training set represented by the set of inputs $x_1 \dots x_n$

Difetti: fully connected -> molte risorse necessarie all'apprendimento, molto tempo necessario se molti neuroni

Utilizzi: approssimazione di funzioni, pattern recognition, classificazione, regression (= trovare un valore discreto ideale in una distribuzione continua)

4. RADIAL BASIS FUNCTION NETWORKS (RBFN)

Struttura:



Tipo: FEED-FORWARD Archi: MONODIREZIONALI

Bias: si, 1 per ogni neurone (ma non contemplato)

Livelli: INPUT+ 1 HIDDEN + OUTPUT (un solo neurone)

F. attivazione:

Gaussian functions (most used):

$$\varphi(r) = \exp\left(-\frac{r^2}{2\sigma^2}\right) \quad \sigma > 0$$

(HIDDEN NEURONS)

σ = spread (scelto da alg. di appr.)

$$r = \|x - t\| \quad t = \text{centro del neurone}$$

$$y = w_1 \varphi_1(\|x - t_1\|) + \dots + w_{m1} \varphi_{m1}(\|x - t_{m1}\|) \quad m1 = \text{numero centri}$$

(OUTPUT NEURON) è lineare

Apprendimento/Correzione dell'errore/Formule: SUPERVISIONATO

1 ALGORITMO: **a)** centri **t** presi casualmente dal training set, tanti quanti sono i neuroni desiderati (< delle istanze di training) **b)** lo spread σ viene scelto dalla normalizzazione

$$\sigma = \frac{\text{Maximum distance between any 2 centers}}{\sqrt{\text{number of centers}}} = \frac{d_{\max}}{\sqrt{m_1}}$$

e dunque l'attivazione hidden diventa:

$$\varphi_i(\|x - t_i\|) = \exp\left(-\frac{m_1}{d_{\max}^2} \|x - t_i\|^2\right)$$

c) per ogni input **x** vogliamo che **y** sia simile a **d** (valore atteso su x).

Φ = matrice delle attivazioni hidden, **w** = vettore pesi hid->out \rightarrow

$$\Phi w = d \quad \text{or} \quad \Phi^T \Phi w = \Phi^T d$$

definiamo la PSEUDO INVERSA come:

$$\Phi^+ = (\Phi^T \Phi)^{-1} \Phi^T$$

e ora possiamo trovare w.

NOTA: se $\Phi^T \Phi$ ha determinante vicino allo 0 o $=0$, si dice che è una matrice singolare, e non è dunque invertibile seppur quadrata

2 ALGORITMO: **a)** centri **t** messi casualmente sulla mappa (tanti quanti sono i neuroni desiderati), tutto il training set viene messo sulla mappa, per ogni input **x** si guarda il t più vicino: $\min \|x - t\|$, si modifica solo quel t: $t(n+1) = t(n) + \eta \|x(n) - t(n)\|$ (n è l'iterazione)

b) lo spread σ viene scelto allo stesso modo di 1 ALGORITMO

c) pesi **w** casuali e aggiustati man mano con delta rule tipo adaline

3 ALGORITMO:

- Apply the gradient descent method for finding centers, spread and weights, by minimizing the (instantaneous) squared error $E = \frac{1}{2}(y(x) - d)^2$
- Update for:
 - centers $\Delta t_j = -\eta_{t_j} \frac{\partial E}{\partial t_j}$
 - spread $\Delta \sigma_j = -\eta_{\sigma_j} \frac{\partial E}{\partial \sigma_j}$
 - weights $\Delta w_{ij} = -\eta_{ij} \frac{\partial E}{\partial w_{ij}}$

Terminazione dell'apprendimento: (1 ALGORITMO) non fa iterazioni, calcola solo pseudo inversa (2 ALGORITMO) quando errore scende sotto soglia / quando viene raggiunto numero di epoche fissato (3 ALGORITMO) uguale a 2 ALGORITMO

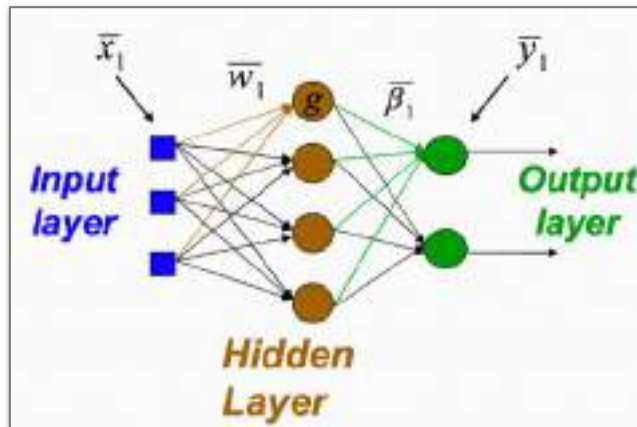
Peculiarità: sempre un solo hidden layer, la funzione di attivazione gaussiana è applicata localmente rispetto al singolo neurone e non alla somma pesata di essi

Difetti: —

Utilizzi: interpolazione approssimata di funzioni continue (dati degli input discreti, cerco di approssimare la loro distribuzione continua)

5. EXTREME LEARNING MACHINES (ELM)

Struttura:



Tipo: FEED-FORWARD Archi: MONODIREZIONALI

Bias: si, sia neuroni hidden che output

Livelli: INPUT+ 1 HIDDEN + OUTPUT

F. attivazione:

$$\varphi(v_j) = \frac{1}{1 + e^{-v_j/\sigma}}$$

$$\text{where } v_j = \sum_i w_{ji} y_i$$

with w_{ji} weight of link from node i

to node j and y_i output of node i == input per il neurone j

(HIDDEN NEURONS)

! viene chiamata $g(x)$ e non $\varphi(x)$

σ = spread

Apprendimento/Formule: SUPERVISIONATO

1 ALGORITMO:

Step 1: Randomly assign input weight w_i and bias b_i ,
 $i = 1, \dots, \hat{N}$.
 Step 2: Calculate the hidden layer output matrix H .
 Step 3: Calculate the output weight β
 $\beta = H^+ T$,
 where $T = [t_1, \dots, t_N]^T$.

$$\begin{bmatrix} g(\bar{x}_1 \cdot \bar{w}_1) & \dots & g(\bar{x}_1 \cdot \bar{w}_N) \\ \vdots & & \vdots \\ g(\bar{x}_N \cdot \bar{w}_1) & \dots & g(\bar{x}_N \cdot \bar{w}_N) \end{bmatrix} \begin{bmatrix} \beta_{11} & \dots & \beta_{1n} \\ \vdots & & \vdots \\ \beta_{N1} & \dots & \beta_{Nn} \end{bmatrix} = \begin{bmatrix} y_{11} & \dots & y_{1n} \\ \vdots & & \vdots \\ y_{N1} & \dots & y_{Nn} \end{bmatrix}$$

$$H \cdot \beta = y \quad \longrightarrow \quad \beta = H^{-1} \cdot y$$

But we want to solve a different problem:

$$H \cdot \beta \approx T \quad \longrightarrow \quad \beta = H^+ \cdot T$$

NOTA: come nelle RBFN anche qui se HT^*H (prima si chiamava $\Phi^T \Phi$) è prossima alla singolarità, non possiamo applicare l'algoritmo per la pseudo inversione. Per risolvere: si aggiunge una costante λ detta termine di regolarizzazione, che (moltiplicata per una matrice identità I grande quanto HT^*H) si somma a HT^*H

pertanto:

$$H^+ = (H^T H + \lambda I)^{-1} H^T$$

2 ALGORITMO:

A different approach consists in evaluating the *singular value decomposition* (SVD) of H :

$$H = USV^T$$

U e V^T sono matrici unitarie: U è $n \times n$, V^T è $m \times m$. S è una matrice con tutti zeri a parte la diagonale principale che contiene i valori discendenti > 0 a partire da in alto a sx. (MA QUALI SONO STI VALORI IN STE MATRICI? BOOOOOO)

Using this strategy, the pseudoinverse matrix H^+ is defined as

$$H^+ = V\Sigma^+U^T$$

Σ^+ si ricava a partire da S , è come S ma sulla diagonale principale ha i valori (1/valore nella stessa posizione in S)

Terminazione dell'apprendimento: l'apprendimento non è iterativo (tipo 1 ALGORITMO di RBFN)

Peculiarità: teorema di interpolazione: dati N pattern in input e N neuroni hidden con pesi inizializzati casualmente, è sempre possibile trovare dei pesi β (hid->out) per cui vale che $H\beta = T$ (T = insieme dei target)

teorema di approssimazione: (presupposto: non avremo mai N neuroni hidden) risolvendo il sistema con un numero minore di neuroni hidden, grazie alla pseudo-inversa H^+ si ottiene comunque la soluzione con una certa tolleranza

- il training è anche più di 100 volte più veloce rispetto ad altri modelli!

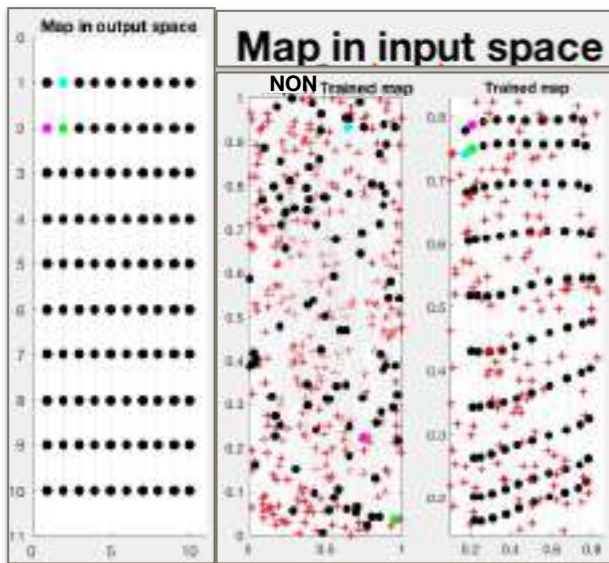
Difetti: anche utilizzando SVD (2 ALGORITMO) se si incontrano matrici singolari si avranno problemi —> la regolarizzazione è comunque necessaria!

con molti dati da trattare si preferisce comunque utilizzare un sistema con discesa del gradiente

Utilizzi: —

6. SELF-ORGANIZING MAPS (SOM)

Struttura:



OUTPUT SPACE: bidimensionale, è la mappa vera e propria dove ogni neurone ha associato un vettore pesi, che cambiano in base alla vicinanza, nell'*input space*, del neurone con i suoi vicini dell'*output space*

INPUT SPACE: qualsiasi dimensione (nell'esempio è bidim.), è la mappa dove vengono posizionati i dati del training set (+) e dove i neuroni si spostano per ricoprirla partendo da posizioni casuali e risistemandosi similmente all'*output space*, grazie alla *neighborhood*

(i numeri sul grafico non sono importanti)

Tipo: FEED-FORWARD Archi: SIMMETRICI Bias: no

Livelli: è più corretto parlare di "spazi": INPUT SPACE, OUTPUT SPACE

F. attivazione:

lineare: $Y = w_{i1}x(1) + \dots + w_{in}x(n)$ i =neurone, n =dimensione input

ma in pratica questa funzione si traduce nel calcolo della distanza del neurone i dall'input x

Apprendimento/Formule:

NON SUPERVISIONATO, COMPETITIVO (=neuroni "vincono" su altri neuroni a ogni iterazione)

-Il neurone con Y maggiore (= con distanza minore) per ogni input x diventa la BMU (Best Matching Unit) per quell'input.

Once individuated the BMU i for input x , his weight vector w_i is modified towards x . In this way, the euclidean distance between x and i 's weights is diminished and the following times that x or a similar input will be presented to the net, the BMU will have chances of being i .

Simplifying, the weights of i , BMU for input x , are updated as follows:

$$w_i(n+1) = w_i(n) + \eta(n)(x - w_i(n))$$

$\begin{matrix} x & & w_i \end{matrix}$

$Ex: [0.6 \ 0.9] = [0.2 \ 1] + 0.5([1 \ 0.8] - [0.2 \ 1])$

After correction, w_i is more similar to x than before
 If $\eta = 1$ w_i becomes equal to x

- The learning rate usually decreases with time. In general, the decrease of learning rate is: $\eta(n) = \eta_0 \exp(-\frac{n}{\tau})$

<— evita overfitting su trainingset

Ma non si "sposta" solo la BMU = non cambiano solo i pesi della BMU, anche quelli dei neuroni vicini (sull'OUTPUT SPACE).

La vicinanza è definita dalla funzione **Neighborhood** $h_{ji}(n) = \exp\left(-\frac{d_{ji}^2}{2\sigma(n)^2}\right)$ i =BMU, j =altri neuroni, σ =spread $\sigma(n) = \sigma_0 \exp\left(-\frac{n}{\tau}\right)$ questo parametro diminuisce con il passare delle iterazioni (n), restringendo di conseguenza la curva gaussiana della Neighborhood, d^2_{ji} =distanza quadratica tra j e i

Man mano che la Neighborhood si restringe, sempre meno neuroni j vengono considerati "vicini" della BMU, e dunque smetteranno di spostarsi insieme alla BMU.


Terminazione dell'apprendimento: "quando la mappa è abbastanza ben distribuita sul training set nell'INPUT SPACE, ovvero quando è ben ordinata topologicamente". Ecco come è diviso l'apprendimento:

(1) Auto-organization: In this phase the topological ordering appears. It can require 1000 iterations. Usually:

- learning rate η decreases from .1 to 0.01 and
- neighborhood function h_{ji} at the beginning can include almost all the neurons and shrink until it includes only i

(2) Convergence phase: In this phase the correspondence between neuron weights and inputs becomes more precise. Usually at least 500 iterations:

- learning rate η around 0.01 (always > 0)
- neighborhood function h_{ji} contains only i



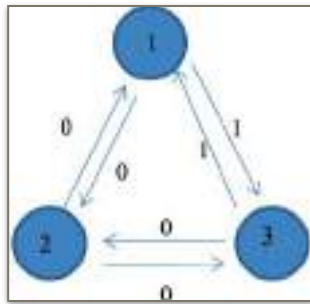
Peculiarità: principio delle SOM è simile alle mappe topologiche che crea il cervello, pertanto le applicazioni di esse possono essere moltissime

Difetti: —

Utilizzi: classificazione / clustering

7. RETI DI HOPFIELD (HOPFIELD NETWORKS)

Struttura:



I neuroni 1, 2, 3, sono solo numerati come tali. In una rete di hopfield i **neuroni sono binari**.

Tipo: RECURRENT Archi: SIMMETRICI Bias: si, ma non considerato

Livelli: 1 LIVELLO FULLY CONNECTED → N neuroni → (N-1)*N archi / pesi w , $w_{i,j} = w_{j,i}$

F. attivazione:

ATTIVAZIONE
BINARIA DEL
SINGOLO
NEURONE

$$v_j = \sum_{i=0}^N w_{ji} y_i(n)$$

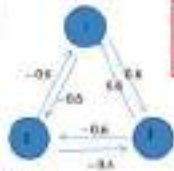
valore del neurone ad inizio iterazione n

$$\varphi(v_j)(n) = \begin{cases} 1 & \text{if } v_j(n) > 0 \\ -1 & \text{if } v_j(n) < 0 \\ \varphi(v_j)(n-1) & \text{if } v_j(n) = 0 \end{cases}$$

stato stabile

Apprendimento/Correzione dell'errore/Formule: NON SUPERVISIONATO

- The states are stables thanks to weights.
- For instance $[1, -1, 1]$



We adjust the weights so that the informations to memorize become stable states

Intuitively, a state is stable if

- Units with the same activation are connected with positive weights
- Units with opposite activations are connected with negative weights

- For this reason, also $[-1, 1, -1]$ is stable (in general, if x is a stable state, also $-x$ is a stable state)

$$E - E' = -\frac{1}{2} \left((2 \sum_{j \neq k} w_{kj} y_k y_j - 2 \sum_{j \neq k} w_{kj} y'_k y_j) \right)$$

$$E - E' = -\sum_j w_{ij} y_j (y_k - y'_k)$$

We distinguish 2 cases:

- y_k passes from $+1$ to -1 . Hence $y_k - y'_k > 0$ and $\sum_j w_{ij} y_j < 0$. Then

$$-\sum_j w_{ij} y_j (y_k - y'_k) > 0 \text{ and } E > E'$$

- y_k passes from -1 to $+1$. Hence $y_k - y'_k < 0$ and $\sum_j w_{ij} y_j > 0$. Then again

$$-\sum_j w_{ij} y_j (y_k - y'_k) > 0 \text{ and } E > E'$$

At each change E lowers. Since the states are finite, at a given point we get to a state that does not change: a stable state

- Convergence Theorem:** Given any state of the network, the network converges to a stable state.
 - Given N units, there are 2^N possible states of the network
 - To each of these states is associated an **Energy** value
 - We prove that each change of state of the network leads to a lowering of the Energy.
 - There is a moment in which nothing can change any further, there are no other states to move in.

$$\text{Energy function } E = -\frac{1}{2} \sum_i \sum_j w_{ij} y_i y_j$$

Each product $w_{ij} y_i y_j$ is positive if y_i and y_j have the same sign and w_{ij} is positive or they have opposite sign, and w_{ij} is negative

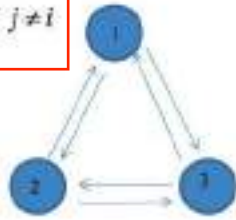
The energy E measures the harmony of the actual state with the fundamental memories (the bigger E , the lower the harmony) and the propension to change state

We prove that each change in state diminishes E . ($E > E'$)

PROCESSO DI AUTO
STABILIZZAZIONE:
(partendo da uno stato
qualsiasi viene raggiunta
sempre la *stabilità*=
un minimo di energia)

• Given M fundamental memories, f_1, \dots, f_n , vectors of dimension N , we memorize them by letting:

$$w_{ji} = \frac{1}{M} \sum_{k=1}^M f_k(i) \cdot f_k(j) \text{ if } j \neq i$$



PROCESSO DI MEMORIZZAZIONE:

(se diamo alla rete un input adatto alle sue dimensioni esso diventerà una *memoria fondamentale*, ovvero uno stato stabile che la rete sarà in grado di ricreare quasi certamente (vedi *Difetti*) partendo da un input qualsiasi, grazie ai valori sugli archi simmetrici)

If we want two fundamental memories: $[1, -1, 1]$ and $[-1, 1, -1]$

$$w_{21} = w_{12} = \frac{1}{2} \sum_{k=1}^2 f_k(1) \cdot f_k(2) = \frac{1}{2} (1 \cdot (-1) + (-1) \cdot (1)) = -1$$

$$w_{23} = w_{32} = \frac{1}{2} \sum_{k=1}^2 f_k(2) \cdot f_k(3) = \frac{1}{2} \cdot (1 \cdot (-1) + (-1) \cdot (1)) = -1$$

$$w_{31} = w_{13} = \frac{1}{2} \sum_{k=1}^2 f_k(1) \cdot f_k(3) = \frac{1}{2} (1 \cdot 1 + (-1) \cdot (-1)) = 1$$

Terminazione dell'apprendimento: raggiungimento di uno stato stabile (= di un livello di energia minima). Non sempre ciò corrisponde al raggiungere una memoria fondamentale.

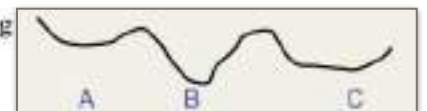
Peculiarità:

- They complete partial patterns
- They generalize: given an input similar to what has been memorized, they recover the corresponding information
- They are fault tolerant: if some synapses get broken (brain damage) the output is still reasonable
- They allow the extraction of prototypes: if the network learns several similar informations, it creates their prototype and this is the explicitly presented state.
- The learning rule is Hebb like, which is biologically plausible and there is evidence that it exists in the brain
- The networks can account for context effects: we remember better what is learned if we are put in the same context

Difetti:

- Unfortunately, not all the stable states are fundamental memories memorized during storage. There are spurious states:
- The opposite of a stable state is a stable state
- Combinations of stable states are stable states
- Not all the fundamental memories are stable states
- Storage capacity with few errors given N units = $0.14 N$
- With almost no error: $\frac{N}{2 \log N}$

E



W

A, B, C sono punti di minimo locale
B è anche il minimo globale

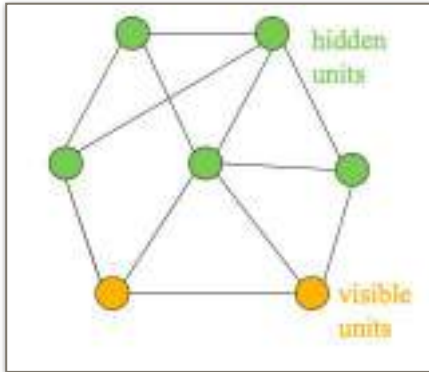
Se la configurazione iniziale dei pesi W è vicina a quella per raggiungere A oppure C, la rete di Hopfield non sarà in grado di arrivare a B, perchè ogni cambiamento di pesi equivale a una discesa di energia
"non potrà scavalcare le colline"

Utilizzi: pattern completion, generalizzazione

8. MACCHINE DI BOLTZMANN E BOLTZMANN RESTRICTED (RBM)

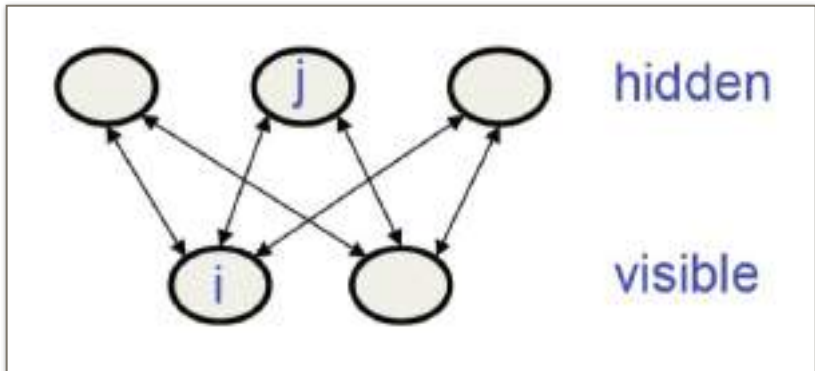
Struttura:

MACCHINA DI BOLTZMANN



1+ livelli hidden, connessioni anche tra neuroni dello stesso livello.

RESTRICTED BOLTZMANN MACHINE



Un solo livello hidden, connessioni solo da un livello all'altro.

In un modello di boltzmann i **neuroni sono binari e stocastici**.

Tipo: RECURRENT Archi: SIMMETRICI Bias: si, ma non considerato

Livelli: VISIBLE + n HIDDEN (MACCHINA DI BOLTZMANN) , VISIBLE + HIDDEN (RBM)

F. attivazione:

$$p(s_i=1) = \frac{1}{1 + e^{-\Delta E_i/T}}$$

-I neuroni sono stocastici: viene misurata la probabilità che lo stato del neurone i assuma valore 1.

$p(s_i=1) > 0.5$ && $s_i=1 \rightarrow$, E scende

$p(s_i=1) < 0.5$ && $s_i=0 \rightarrow$ E scende

$p(s_i=1) = 0.5 \rightarrow$ E non varia in ogni caso

T =temperatura , se alta permette di "scavalcare le colline" che separano i minimi, e arrivare più probabilmente al minimo globale. Noi l'abbiamo sempre considerata costante ($= 1$)

$$\text{Energy gap} = \Delta E_i = E(s_i=0) - E(s_i=1) = b_i + \sum_j s_j w_{ij}$$

b =bias per i . Non lo consideriamo ($= 0$)

Apprendimento/Correzione dell'errore/Formule: NON SUPERVISIONATO

obiettivo dell'apprendimento è massimizzare la $\log p(v)$ assegnata a ogni training vector che il sistema deve imparare a generare/riconoscere.

$$\frac{\partial \log p(v)}{\partial w_{ij}} = \langle s_i s_j \rangle_v - \langle s_i s_j \rangle_{model}$$

derivata parziale della $\log p(v)$ rispetto ad un peso della rete. v è un training vector

prodotto degli stati dei neuroni i e j quando v è fissato ("clamped") sulle unità visibile

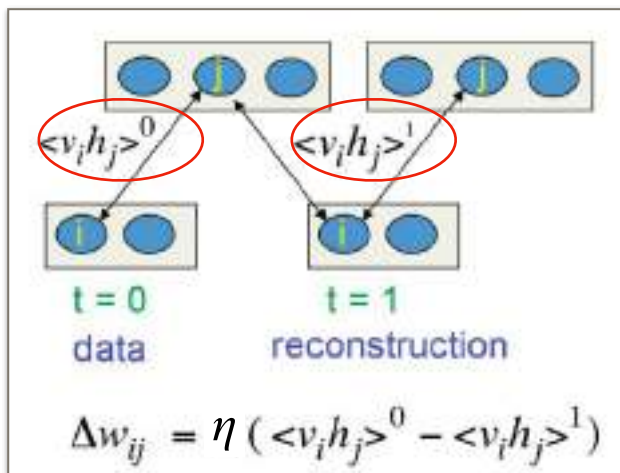
prodotto degli stati dei neuroni i e j senza nulla di fissato ("free")

Il training di un modello di boltzmann ha fase
1-**clamped**: training vector fissato alle unità visibile \rightarrow trovo i valori degli stati delle unità hidden
2-**free**: i valori delle unità visibile ora possono cambiare, lascio operare la rete perchè raggiunga l'equilibrio

MACCHINA DI BOLTZMANN: le due fasi sono molto costose a causa delle connessioni intralivello, la rete effettua moltissimi calcoli per trovare l'equilibrio.

RBM: [fase clamped] una sola iterazione, non ci sono connessioni intralivello

[fase free] **contrastive divergence**: è stato osservato empiricamente che facendo effettuare solamente 4 iterazioni...



1: i dati sono sulle unità visibili

2: ricalcolo i valori delle hidden e di

conseguenza $\langle v_i h_j \rangle^0$ per ogni coppia i, j

3: **reconstruction** dei dati interpretati sulle visibili

4: ricalcolo i valori delle hidden e di

conseguenza $\langle v_i h_j \rangle^1$ per ogni coppia i, j

—>

ora posso calcolare il Δw per ogni peso w_{ij} !

η = learning rate

...la rete lavora meglio di quanto farebbe se la lasciassimo "fantasticare" finché non trova il suo equilibrio seguendo la discesa del gradiente della log likelihood.

Terminazione dell'apprendimento: al termine di contrastive divergence per ogni training vector che si desidera immagazzinare

Peculiarità: (modelli di boltzmann) oltrepassano il problema del massimo numero di memorie fondamentali immagazzinabili da una rete di hopfield
(RBM) sono molto utili per l'inizializzazione dei pesi delle DNN

Difetti: (MACCHINE DI BOLTZMANN) l'apprendimento richiede costi enormi in termini di risorse e potenza di calcolo

Utilizzi: (RBM) inizializzazione dei pesi delle DNN