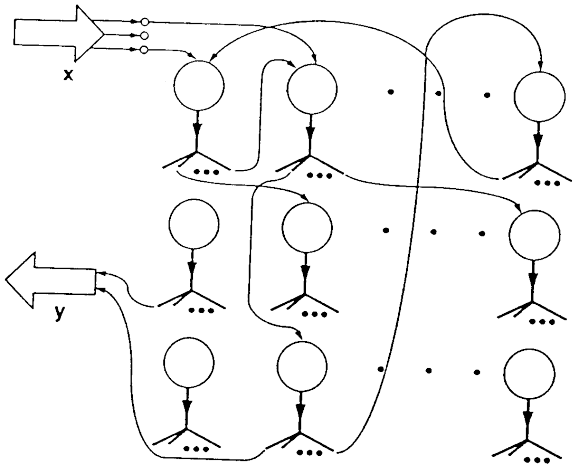
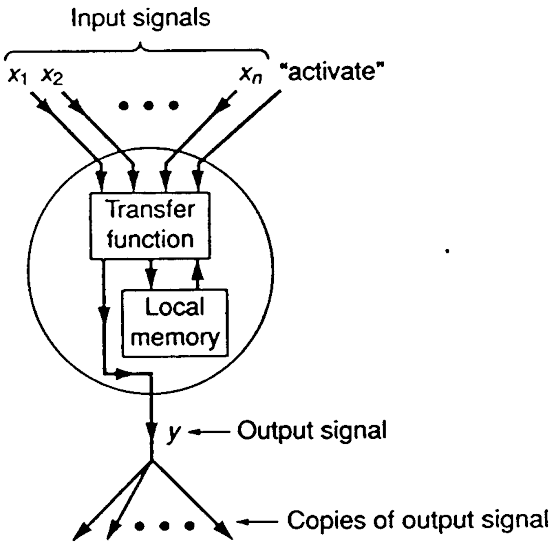
Consideriamo la definizione di rete neurale (RN) secondo Hecht-Nielsen. Una *Rete Neurale è* una struttura parallela che processa informazioni distribuite. L'idea di partenza è di immediata comprensione: si tratta di collegare insieme elementi in grado di eseguire compiti semplici, in modo che l'insieme risultante permetta di eseguire compiti complessi. Questo è proprio quanto avviene nella mente umana, costituita da una intricata rete di elementi relativamente semplici detti appunto neuroni. Una RN consta di elementi di elaborazione, detti ancora neuroni (o PEs), che possono avere una memoria locale e che sono in grado di processare localmente informazioni ricevute in ingresso. L'informazione contenuta in ciascun neurone è poca cosa, e vedremo che in certi casi un singolo neurone può essere del tutto trascurato senza ripercussioni sul contenuto informativo dell'intera RN.

I neuroni sono interconnessi tramite canali (detti *connessioni*) che trasmettono segnali unidirezionali. Ogni neurone ha una singola connessione di uscita che si dirama in un certo numero di connessioni collaterali; ognuna di questa trasporta lo stesso segnale, il segnale d'uscita del neurone. Questo segnale d' uscita può essere di qualunque tipo matematico. La computazione compiuta all'interno di ciascun neurone può essere definita arbitrariamente con l'unica restrizione che deve essere completamente locale; cioè deve dipendere solo dai valori correnti dei segnali d'ingresso che arrivano al neurone tramite opportune connessioni e dai valori immagazzinati nella memoria locale del neurone.

Lo schema a lato raffigura una RN. I cerchietti rappresentano i neuroni. Come si vede, alcuni dei neuroni ricevono in ingresso un vettore x. Ogni neurone produce una uscita, che può diventare l'ingresso di un altro neurone, oppure essere parte dell'uscita complessiva y (ancora un vettore) della rete. Non c'è correlazione tra la dimensione di x e quella di y.





A sinistra viene indicata la struttura tipica di un neurone. L'insieme dei segnali di ingresso è $x_1, \ldots, x_n$. Il neurone realizza, a partire da tali valori e dai dati memorizzati nella memoria locale, una funzione di trasferimento. Detta funzione oltre a determinare l'uscita del neurone può modificarne la memoria locale. Questo perché i neuroni attraversano, come si è detto, anche una fase di addestramento; la memoria locale conterrà la conoscenza maturata durante tale fase.

Quando il neurone è stato addestrato, entra nella fase operativa, in cui riceve in ingresso dei segnali ed elabora di conseguenza l'uscita. Nella

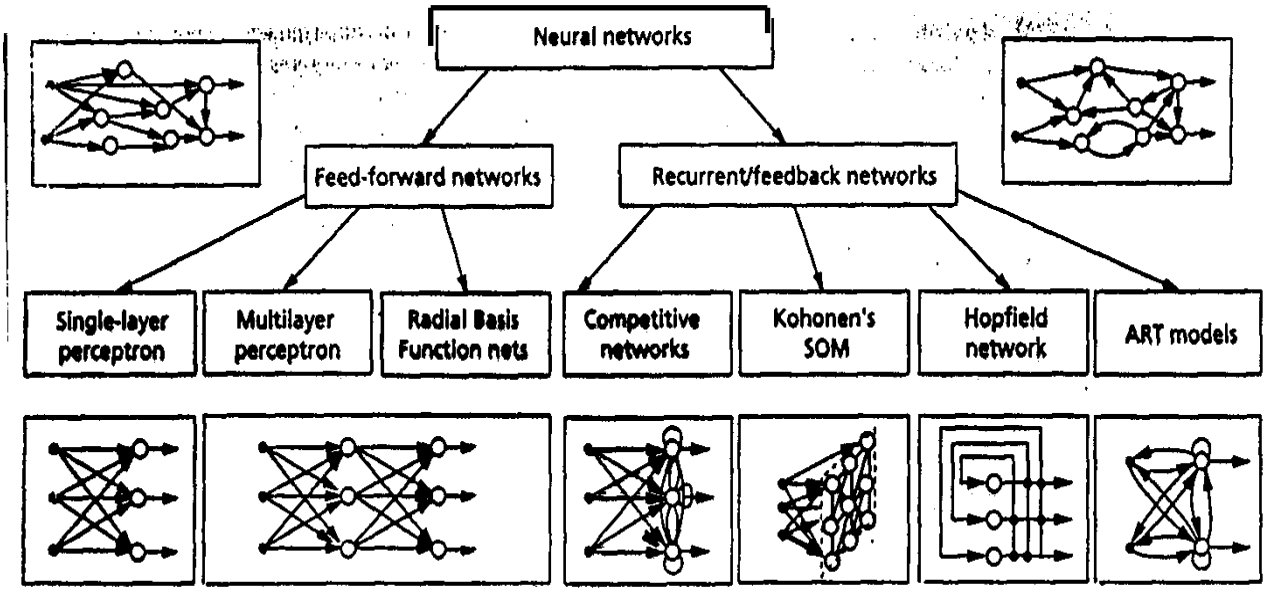
fase operativa la memoria locale non viene più modificata (alcuni modelli di RN fanno eccezione a questa regola: *apprendimento adattativo*, nel quale l'apprendimento non termina mai).

In questo corso prendiamo in esame esclusivamente RN di tipo software. Per ogni neurone va specificata la funzione di trasferimento realizzata. Dovremo esaminare inoltre le varie architetture disponibili nonché le leggi di apprendimento che vengono di volta in volta applicate.

Il disegno che segue propone alcune tipiche topologie di RN. I modelli possono essere suddivisi in due grandi categorie: le reti *feed-forward* e quelle *ricorrenti*. Le reti **feed-forward** sono caratterizzate da un unico flusso di dati unidirezionale, privo cioè di cicli. Si riconoscono in essi degli strati o layer; abbiamo per esempio modelli a singolo strato di percettroni.

Altri modelli di questa famiglia si presentano con una configurazione a multistrato (MLP, dette anche impropriamente reti Back-Propagation), in cui ciascuno strato è collegato al successivo. Il primo strato della rete riceve in parallelo l'ingresso; elabora quindi l'uscita, che sarà l'ingresso dello strato successivo, e via dicendo. Questi modelli sono di gran lunga i più noti e diffusi.

I modelli Radial Basis Function sono caratterizzate da una speciale modalità di addestramento e da una diversa funzione di trasferimento per il singolo neurone. Noi non tratteremo questi modelli.



La seconda grande famiglia tassonomica di RN è quella che comprende le **reti a connessione laterale** e le **reti ricorrenti**, dette anche feedback networks.

Esempi di reti a connessione laterale sono le reti competitive nelle quali ogni neurone, oltre a dare un contributo verso l'uscita, è collegato ai neuroni dello stesso strato. Possiamo avere anche in questo caso una stratificazione singola o multipla.

Abbiamo anche le mappe auto-organizzanti di Kohonen (SOM), la differenza topologica consiste in una configurazione a griglia: ogni neurone è collegato solo ad un certo numero di altri (es: ai 4 o agli 8 che gli stanno intorno) e non a tutti i neuroni del medesimo strato, come nel caso precedente.

Nelle reti ricorrenti l'uscita di ciascun neurone viene riproposta in ingresso al neurone stesso. Un esemplare di questo modello è dato dalle reti di Hopfield, utili nei problemi di ottimizzazione. Nei modelli ART si riscontra la compresenza del feedback e delle connessioni laterali.

Esistono diverse **modalità di addestramento**. L'addestramento avviene sempre grazie ad un certo numero di esempi prelevati dal mondo reale. Nell'addestramento supervisionato, gli esempi forniti alla RN sono delle coppie (*ingresso*, *uscita corrispondente desiderata*). Per capire il punto, possiamo anticipare il fatto che un utilizzo tipico delle RN è quelle dei *classificatori*: RN che

devono essere in grado di riconoscere oggetti appartenenti a diverse categorie. La descrizione viene fatta mediante vettori di numeri reali; a fronte di un certo ingresso vettoriale, la RN si deve esprimere attribuendolo ad una classe. Nella modalità supervisionata, diremo alla rete che, quando riceve un ingresso ben preciso, deve attribuirlo alla classe X. Altra classe di problemi risolti dalle RN è l'inseguimento (approssimazione) di funzioni. Una funzione in una o più variabili è nota per punti (non ne conosciamo la forma analitica); la RN deve riuscire ad approssimarla nel modo migliore possibile, calcolandone il valore anche in punti diversi da quelli proposti in ingresso. Nella fase di addestramento, diremo alla RN quale valore di uscita corrisponde a ciascun ingresso assegnato. Quanto nella modalità operativa riceverà un ingresso diverso, la rete dovrà tirare fuori un valore approssimato dell'uscita quanto più vicino possibile a quello reale.

Nell'addestramento non supervisionato forniamo alla rete dei soli valori di ingresso (il TS, training set) senza precisare le relative uscite. La rete è in grado di auto-organizzarsi (come nelle SOM cui si è accennato prima) modificando la propria conoscenza interna (ovvero la memoria locale dei neuroni).

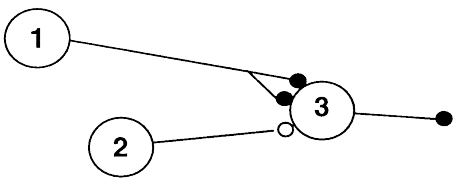
Fra questi due estremi c'è l'addestramento graded: tipicamente la rete evolve in maniera autonoma, come nella modalità non supervisionata. I particolari istanti di tempo viene però utilizzata anche l'informazione sull'uscita desiderata. Si parla anche di reinforcement: ad intervalli di cicli di addestramento scanditi da un periodo si affianca alla modalità non supervisionata una modalità supervisionata. Noi non considereremo questo genere di addestramento.

Esistono cinque categorie fondamentali di **leggi di apprendimento**. Nel *performance learning* l'obbiettivo è di massimizzare o minimizzare una certa funzione, in particolare un certo indice prestazionale. Tutti i neuroni partecipano in ugual misura dell'addestramento. Esempi tipici di reti che rientrano in questa categoria sono le reti a percettroni.

Nel *competitive learning* invece durante l'apprendimento si genera una competizione tra i diversi neuroni. Ad ogni passo dell'addestramento solo una parte dei neuroni (che 'vince' la competizione) modifica la propria memoria (stato interno). Questo è quanto avviene ad esempio nelle SOM di Kohonen.

Non considereremo in questo corso le altre tre categorie di leggi di apprendimento, che citiamo per completezza: *coincidence learning, filter learning, spatiotemporal learning*. Esistono anche esempi di reti nelle quali non esiste una vera e propria fase di apprendimento (reti di Hofield).

Non c'è perfetta ortogonalità tra le modalità di addestramento e le leggi di apprendimento: in molti casi alle leggi di apprendimento possono essere applicate modalità di addestramento supervisionate o non supervisionate, in altri (es. back propagation) la sola modalità superivisionata.

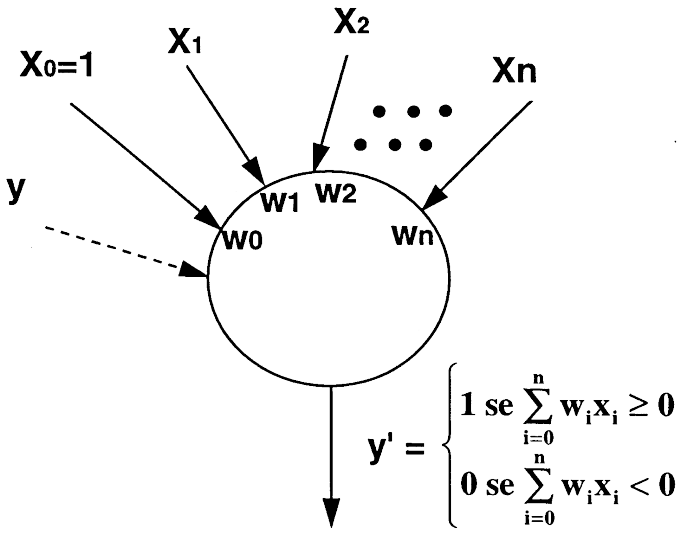


N3(t) = N1(t-1) & (not N2(t-1))

Un progenitore delle RN: IL MODELLO DI MCCULLOCH-PITTS (1943). In questo prototipo non si dovrebbe nemmeno parlare, a rigore, di neuroni; i 'cerchietti' del disegno a lato si limitano a calcolare delle funzioni logiche come quella che si è indicata. Il neurone all'istante t calcola una AND fra le uscite dei neuroni agli istanti precedenti. Uscite e ingressi di ciascun 'neurone' possono essere eventualmente negati (pallino bianco).

IL PERCETTRONE DI ROSENBLATT (1957). È il primo vero e proprio modello di RN, giacchè prevede una sia pur rudimentale legge di apprendimento sulla base di esempi. Il percettrone di Rosenblatt calamitò l'attenzione degli appassionati di AI per oltre un decennio, quando verso la fine degli anni 1960 ci si rese conto che tale modello non poteva risolvere che problemi molto semplici rispetto alle reali necessità applicative. Tuttavia verso la metà degli anni 1980 alcuni

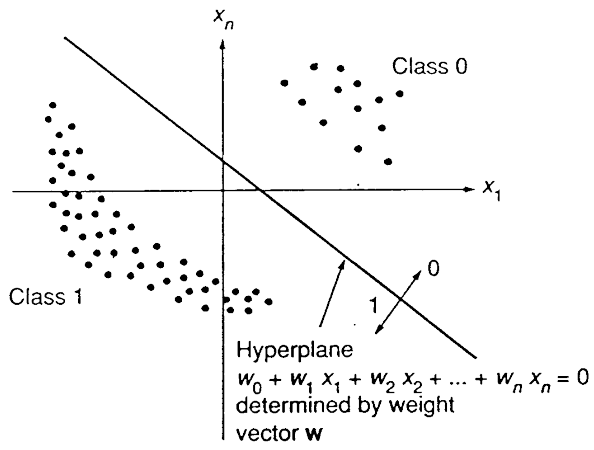
studiosi americani riuscirono a superare i problemi legati a questa semplice struttura, il che rappresentò il vero punto di partenza verso lo sviluppo recente delle RN.



Si ha una serie di ingressi X_1, …, X_n. più l'ulteriore ingresso X_0 che è fisso al valore 1. La memoria locale del neurone è rappresentata dai valori w_0, w_1, …, w_n, detti PESI. A ciascun ingresso è cioè associato un peso, che è un valore reale. La funzione di trasferimento è semplicemente una somma pesata degli ingressi moltiplicati per i pesi (in pratica un prodotto scalare tra vettori). L'uscita del percettrone è a livelli e, per l'esattezza, binaria. Se i prodotto scalare è non negativo, l'uscita del percettrone è alta, altrimenti bassa. Questo modello si presta bene ad essere usato per problemi di classificazione a due classi soltanto o anche per applicazioni relativi a funzioni logiche, con ingressi e uscite binari.

La fase di addestramento è *supervisionata*. Vengono mostrati al percettrone degli ingressi e le rispettive uscite desiderate y, e in base a ciò il percettrone modifica i pesi in accordo alla legge di apprendimento che prenderemo in esame tra breve.

Consideriamo una interpretazione geometrica della funzione di trasferimento realizzata dal percettrone. La rappresentazione del grafico che segue è bidimensionale: il percettrone riceve due ingressi, x_1 e x_2, e fornisce una uscita binaria. In generale avremo un iperpiano la cui espressione è indicata in calce al diagramma; nel caso bidimensionale l'iperpiano diviene una retta, $w_0 + w_1x_1 + w_2x_2 = 0$. Tale retta delimita le '**regioni di decisione**' del percettrone. Dopo aver ricevuto l'ingresso (x_1, x_2), il percettrone calcola il valore della funzione di trasferimento emettendo in uscita il valore 1 o 0. La retta è il luogo dei punti per i quali il prodotto scalare di pesi e ingressi (più il peso w_0, cui corrisponde l'ingresso fisso unitario), ovvero il valore di uscita del neurone, è nullo. Poiché la retta suddivide il piano in due semipiani, in base all'espressione della funzione di trasferimento del percettrone possiamo concludere che tutti gli ingressi di coordinate (x_1, x_2) del semipiano inferiore sono quelli cui corrisponde uscita bassa, e viceversa dicasi per il semipiano superiore.



Quindi, dal punto di vista geometrico il percettrone opera una separazione lineare dei punti dello spazio (nell'esempio uno spazio a due dimensioni. Nel caso tridimensionale il percettrone genera un piano che divide lo spazio in due semispazi. Nel caso n-dimensionale per n > 3 diventa difficile visualizzare graficamente la situazione). Modificare i pesi w_1, w_2 della funzione di trasferimento significa modificarne la pendenza (*rotazione*); in più la presenza di un termine w_0 fa sì che la retta non sia vincolata a passare per l'origine, e ne regola la *traslazione*. Lo scopo del percettrone (e quindi della sua legge di apprendimento, che è di tipo performance) è la minimizzazione dell'errore.

$$W^{new} = W^{old} + (y-y')\,x$$

Abbiamo riportato a sinistra una semplice legge di apprendimento, che per il momento non giustifichiamo, consistente nel modificare il vettore w dei pesi in modo che il nuovo valore di w sia pari al vecchio, più un termine dato dal prodotto fra l'ingresso vettoriale x e la differenza fra uscita desiderata e uscita del percettrone. Le due uscite y e y' possono valere 1 o 0. Si deduce immediatamente che:

1. se l'uscita desiderata è uguale all'uscita del percettrone, il vettore dei pesi rimane immutato; non è necessario in questo caso modificare la conoscenza interna, ovvero i pesi;
2. se le uscite sono diverse, l'ingresso può essere o sottratto o sommato al vecchio valore del vettore w.

Un esempio: la funzione OR. Supponiamo che il percettrone debba apprendere la funzione booleana or inclusivo, la cui tabella di verità è ricordata a lato. Vogliamo quindi un percettrone che riceva in ingresso due valori binari e restituisca in uscita la or tra i due ingressi. La novità concettuale rispetto alle implementazioni considerate in altre materie di questo corso di laurea sta nel fatto che, stavolta, abbiamo a che fare con uno strumento in grado di apprendere autonomamente, per esempi, il comportamento desiderato.

x_1	x_2	y
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

Tutte le leggi di addestramento partono da una condizione iniziale; i pesi del neurone devono cioè possedere un valore di partenza. Imponiamo come condizione iniziale, per il momento senza giustificare questa scelta, il vettore:

W^0 | 0.5 | 0 | 0 |

Si calcola per ciascun esemplare del TS (che qui coincide con il Data Set) il prodotto scalare di cui alla funzione di trasferimento.

W^0 | 0.5 | 0 | 0 | x | 1 | 0 | 0 |

Si ha quindi 1*0.5 + 0*0 + 0*0 = 0.5. Il percettrone dà valore alto, mentre l'uscita desiderata è zero. È quindi necessario modificare il vettore dei pesi. Applicando la legge di addestramento succitata, si ha il nuovo valore w^1 = (-0.5, 0, 0) (è y = 0, y' = 1).

W^1 | -0.5 | 0 | 0 |

Proponendo il successivo ingresso:

W^1 | -0.5 | 0 | 0 | x | 1 | 0 | 1 |

si ha y' = 0, e quindi un nuovo valore per il vettore dei pesi, a cui va aggiunto il vettore (1, 0, 1) (è y = 1, y' = 0). Rappresentiamo di seguito la nuova coppia (w,x).

W^2	0.5	0	1	x	1	1	0

L'esempio (1,0) viene riconosciuto correttamente (y = y' = 1), e di conseguenza il vettore w^2 non viene modificato (lo scriviamo in nero anziché in rosso). Lo stesso avviene per l'esempio (1,1).

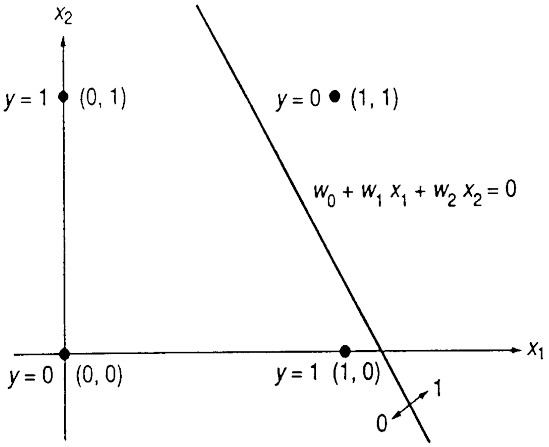
W^2	0.5	0	1	x	1	1	1

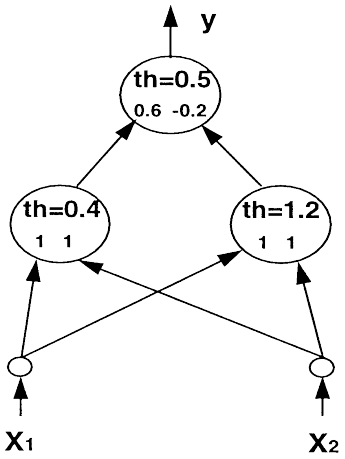
A questo punto, gli esemplari del TS sono riproposti in ingresso al percettrone. Proponiamo una serie di coppie (w, x) relative ai passi successivi dell'apprendimento.

W^2	0.5	0	1	x	1	0	0
W^3	-0.5	0	1	x	1	0	1
W^3	-0.5	0	1	x	1	1	0
W^4	0.5	1	1	x	1	1	1
W^4	0.5	1	1	x	1	0	0
W^5	-0.5	1	1	x	1	0	1
W^5	-0.5	1	1	x	1	1	0
W^5	-0.5	1	1	x	1	1	1
W^5	-0.5	1	1	x	1	0	0

La configurazione giusta dei pesi è quindi (-0.5, 1, 1), giacché tale configurazione non può essere più modificata dagli ingressi. La rete a questo punto è in grado di riconoscere correttamente la funzione OR. Geometricamente, abbiamo ottenuto la retta $x_1 + x_2 = 0.5$; è solo una delle possibili rette che risolvono l'assegnato problema. Per diverse condizioni iniziali abbiamo diverse soluzioni.

Giacché l'unica separazione consentita dal percettrone è di tipo lineare, esistono numerosi esempi nei quali il ricorso al percettrone come separatore binario si rivela inefficace. Famosa a questo riguardo è la critica di Minsky, concernente la funzione logica XOR (OR esclusivo). Essa, come è noto, vale 1 se solo uno dei due ingressi è alto, altrimenti vale 0. I due punti menzionati dovrebbero quindi appartenere al semipiano 1, e gli altri due al semipiano 0. Come si vede dal diagramma a seguire, non esiste alcuna retta in grado di determinare una separazione di questo tipo.

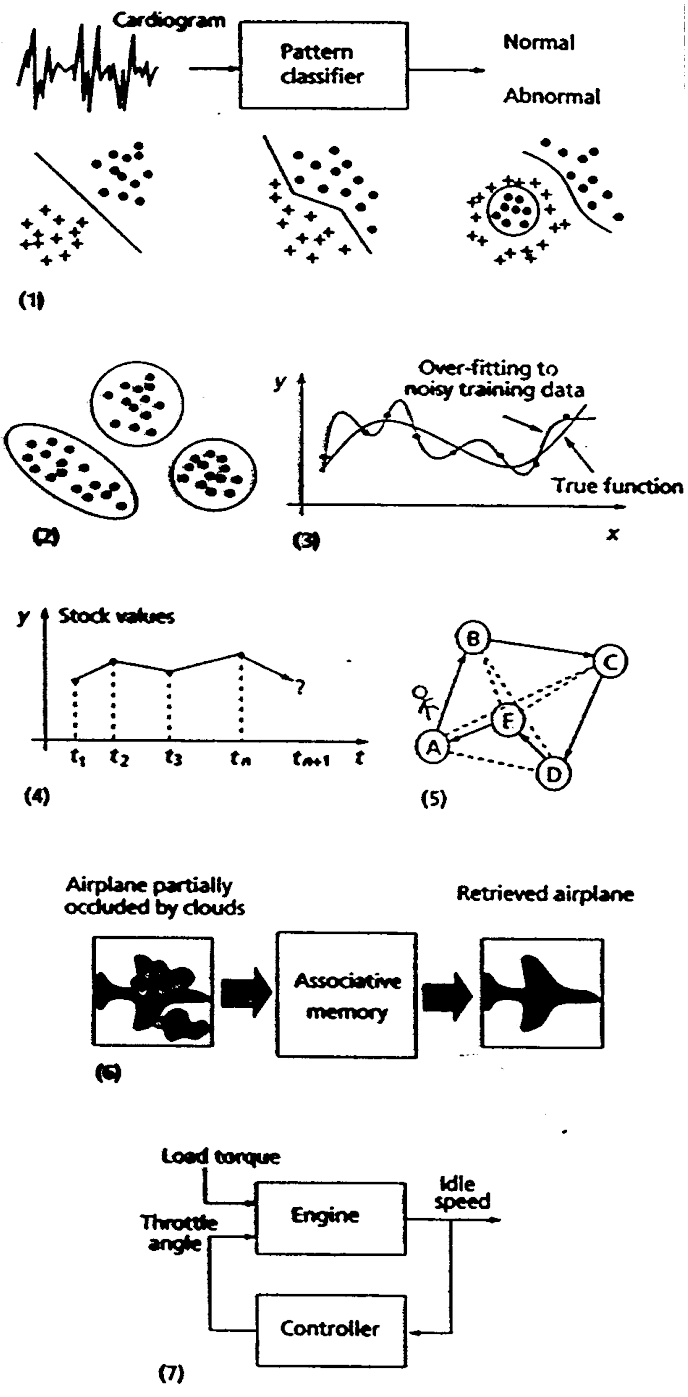




Una possibile soluzione al problema, scoperta nella seconda metà degli anni 1980, è data dal *percettrone multilivello*. Nella figura precedente è indicata una struttura a due livelli. In questo caso è però necessaria una nuova legge di apprendimento, dal momento che, mentre l'uscita desiderata y della rete nel suo complesso (che è poi quella del percettrone superiore) è nota, nulla sappiamo dire sulle uscite desiderate dei due percettroni appartenenti al livello inferiore.

Le RN hanno dimostrato potenzialità significative in vari campi d'applicazione. C'è innanzitutto la **Pattern Classification** (riconoscimento di forme). Una RN agisce come classificatore, discriminando ad esempio una serie di elettrocardiogrammi in ingresso fra normali o indicativi di una patologia. Le reti maggiormente impiegate in tale ambito sono quelle di tipo competitivo e quelle basate su percettroni.

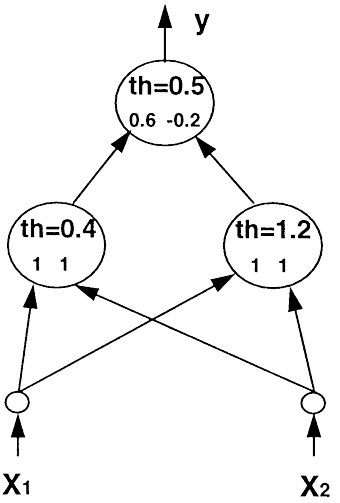
Le mappe auto-organizzanti sono largamente usate nel **Clustering**. Abbiamo un certo numero di esemplari, di cui non si conosce a priori la classe di appartenenza, e non è noto nemmeno il numero delle classi. L'obbiettivo è individuare proprietà comuni che costituiscano la base di una classificazione effettuata a posteriori. Il problema è facilmente visualizzabile in uno spazio bidimensionale o tridimensionale. Potrebbe risultare che in una certa zona del piano vi è un aggregato di oggetti, il che è indicativo del fatto che tali oggetti possiedono proprietà simili. A posteriori associeremo a classi separate gruppi (cluster) di oggetti che risiedono in una stessa parte dello spazio. Le coordinate del piano possono essere in questo caso una misurazione (secondo una certa scala) delle loro proprietà (ad esempio, peso e altezza di un individuo). In uno spazio n-dimensionale, un tale ragionamento geometrico non è più fattibile, per cui si usano appositi algoritmi di raggruppamento. Poiché in casi di questo genere non si conosce a priori il numero di classi, non si può applicare un algoritmo di apprendimento supervisionato.



Abbiamo poi la categoria di **Approssimazione di Funzioni**: è data una funzione definita per punti. La RN apprende la corrispondenza analitica fra x e y. Come è noto, esistono vari strumenti di approssimazione che non hanno nulla a che vedere con l'AI, ma l'approccio neurale risulta più efficiente soprattutto quando si è in presenza di forti non linearità.

Strumenti di Previsione: a partire da (n-1) osservazioni precedenti, la RN ha il compito di predire l'n-sima istanza (*serie storiche*).

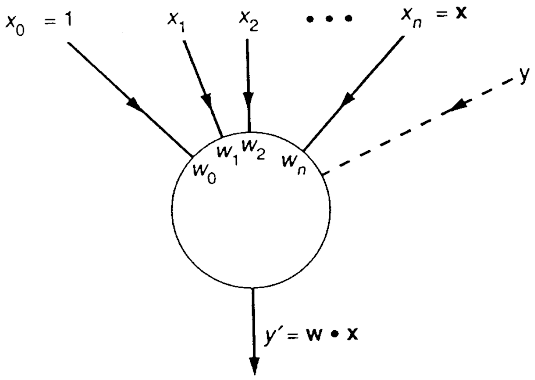
Ottimizzazione: particolari RN (reti ricorrenti di Hopfield) consentono di risolvere problemi classici di ottimizzazione come quello del Commesso Viaggiatore. Queste stesse reti permettono di realizzare inoltre le **Memorie Associative** (dette anche *indirizzabili per contenuto*: il sistema contiene un repertorio di oggetti; sollecitato con un oggetto d'ingresso, restituisce l'oggetto del repertorio che risulta più simile all'ingresso; quest'ultimo sarà ascritto all'eventuale classe di appartenenza dell'oggetto scelto) e di approcciare problemi di **Controllo**.

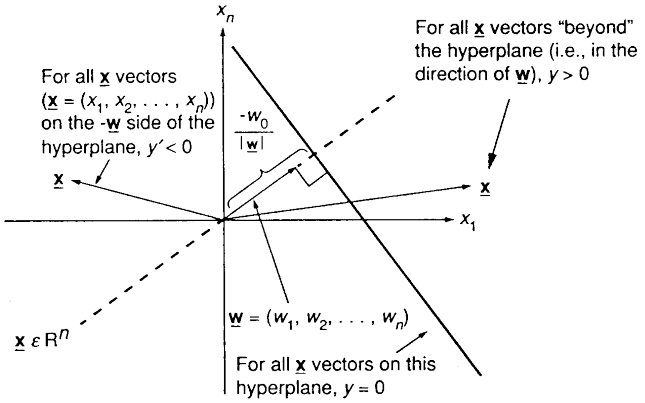


Abbiamo visto la volta scorsa come Minsky e altri criticarono il modello del percettrone, rilevando il fatto che esso non potesse risolvere problemi non linearmente separabili, sebbene molto semplici, come quello della XOR. Possibile soluzione è quella del **Percettrone Multilivello** indicato nel disegno a sinistra. I pallini più piccoli della parte inferiore denotano lo 'strato d'ingresso' del percettrone, che non effettua una vera e propria computazione, ma serve solo per far fluire i dati d'ingresso agli strati successivi.

Per comprendere le scritte come la 'th=0.4' possiamo ricordare la forma della funzione di attivazione del percettrone: da un prodotto scalare si determina una soglia, che definisce il valore binario di y'. La soglia è lo zero. Tuttavia, se estraiamo il peso w_0 dalla sommatoria e lo portiamo a secondo membro, si comprende immediatamente come proprio il peso w_0 possa essere visto come soglia. Nella figura a sinistra th (*threshold)* è appunto la soglia di attivazione del percettrone. Si può verificare (non lo faremo) che questa struttura è effettivamente in grado di risolvere problemi come quello dell'OR esclusivo. Purtroppo rimane in piedi il problema cui si accennava al termine dell'ultima lezione: la legge di apprendimento usata nel modello di Rosenblatt è inapplicabile, dal momento che si conosce l'"uscita desiderata" solamente dell'ultimo strato e non di ogni singolo neurone. Non sapremmo quindi come modificare i pesi di tutti gli altri neuroni. Poco dopo l'individuazione di questo problema Rumhelhart e altri riuscirono tuttavia ad aggirare tale ostacolo, ricavando una legge di apprendimento ed un algoritmo di addestramento adatti per il percettrone multilivello (ci torneremo in una prossima lezione).

Il Prossimo modello che consideriamo è ancora una struttura a singolo neurone, l'ADALINE (ADAptive LiNear Element). La legge di apprendimento usata è quasi identica a quella di Rosenblatt, mentre la differenza sostanziale sta nel fatto che l'uscita y' (funzione di attivazione) è proprio il prodotto scalare tra i vettori w e x: non c'è un concetto di soglia. Il modello è quindi ancora lineare. È sempre presente un peso





w_0 associato alla connessione fittizia $x_0 = 1$. Dal punto di vista geometrico, la retta che si ottiene ha lo stesso significato del caso del percettrone (è il luogo dei punti per cui il prodotto scalare fra il vettore dei pesi e quello degli ingressi risulta nulla). Il valore dell'uscita, che qui è continua e non binaria, è tanto maggiore quanto più ci si allontana da tale retta.

Consideriamo una classica applicazione: l'approssimazione di una funzione y di un certo numero di variabili. All'uopo l'elemento va opportunamente addestrato. Consideriamo allora la seguente espressione per l'errore quadratico medio:

$$\mathbf{F(w)} = \lim_{N\to\infty} \frac{1}{N} \sum_{k=1}^{N} (y_k - y'_k)^2 = E[(y_k - y'_k)^2]$$

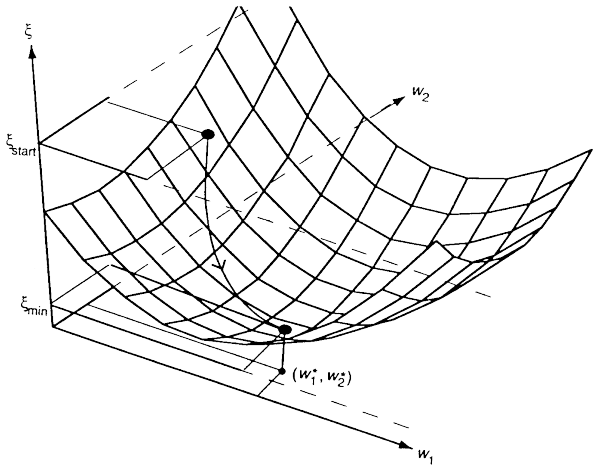
k varia tra 1 a N (numero di elementi del TS). Lo scarto quadrato medio fra l'uscita k-esima desiderata y e quella effettiva y' per tutti i campioni del TS dipende evidentemente dai pesi, dato che vi dipende l'uscita effettiva. L'errore può quindi essere portato quanto più vicino possibile a 0 modificando i pesi. I seguenti passaggi mettono in evidenza tale dipendenza.

$$\begin{aligned}
\mathbf{F(w)} &= E[(y_k - \mathbf{w}^T\mathbf{x}_k)^2] = \\
&= E[(y_k^2 - 2y_k\mathbf{w}^T\mathbf{x}_k - \mathbf{w}^T\mathbf{x}_k\mathbf{x}_k^T\mathbf{w})] = \\
&= E[y_k^2] - 2\mathbf{w}^T E[y_k\mathbf{x}_k] + \mathbf{w}^T E[\mathbf{x}_k\mathbf{x}_k^T]\mathbf{w} = \\
&= \mathbf{p} - 2\mathbf{w}^T\mathbf{q} + \mathbf{w}^T\mathbf{R}\mathbf{w}
\end{aligned}$$

Il problema diventa quindi quello di trovare il punto di minimo della funzione F(w) di errore. Riportiamo di seguito il punto di minimo di F(w), ottenuto azzerando il gradiente.

$$\begin{aligned}
\nabla\mathbf{F(w)} &= \nabla(\mathbf{p} - 2\mathbf{w}^T\mathbf{q} + \mathbf{w}^T\mathbf{R}\mathbf{w}) = \\
&= -2\mathbf{q} + 2\mathbf{R}\mathbf{w} \\
\nabla\mathbf{F(w)} = \mathbf{0} &\Rightarrow \mathbf{w}^* = \mathbf{R}^{-1}\mathbf{q}
\end{aligned}$$

Abbiamo trovato il vettore dei pesi ottimo, ovvero quello che risolve in senso assoluto il nostro problema di apprendimento. Questa costituisce una novità in un panorama come quello della tesi forte dell'AI che appare fatto di soluzioni approssimate e adattative piuttosto che analiticamente esatte. Comunque, nella formula compare l'inversa di una matrice, che potrebbe non essere calcolabile. È da dire anche che a monte del processo di apprendimento potrebbe non essere disponibile l'intero TS, indispensabile per il calcolo di R e q. In casi come questi la formula analitica non ci aiuta e si rende necessario il ricorso ad un algoritmo di apprendimento come quello che introdurremo fra breve.



Si trova che la funzione di errore è un paraboloide (vedi diagramma a seguire), e quindi il punto di minimo è (salvo casi eccezionali) unico. L'esempio considerato è tridimensionale, nel senso che gli ingressi sono due e quindi la funzione di errore dipende da due variabili. Nella figura sono messi in evidenza due punti: quello più in alto corrisponde all'errore quadratico commesso dalla rete in corrispondenza delle sue condizioni iniziali. Quello inferiore w* minimizza la funzione di errore, e, come si vede, è unico.

La legge di apprendimento per questo modello, detta Delta-Rule, fu postulata da Widrow-Hoff. Dal punto di vista geometrico, la soluzione esatta prima trovata corrisponde a partire da un dato punto del paraboloide, e modificare il vettore dei pesi, seguendo così una certa traiettoria sul paraboloide, nella *direzione* della **massima pendenza** e nel *verso discendente*. La *legge di apprendimento* che ne scaturisce consisterebbe nel modificare i pesi di una quantità proporzionale al gradiente. In basso è indicato il calcolo del gradiente. Il problema di tale formula è che in esso compare una media statistica. Il calcolo esatto richiederebbe quindi un numero infinito di elementi del TS, cosa non possibile.

L'ostacolo viene aggirato introducendo una differenza fra *training by epoch* e *training by pattern*. In quest'ultimo caso, per ogni esemplare del TS effettuiamo una modifica dei pesi (come avviene nel percettrone). Nel training by epoch invece viene effettuata dopo ogni 'epoca' (ogni evento di presentazione dell'intero training set al sistema che si vuole addestrare costituisce un'epoca). Nel caso di Adaline a rigore si dovrebbe effettuare un training by epoch. Widrow-Hoff dimostrarono comunque che anche adottando un criterio di training by pattern, cioè calcolando il gradiente campione per campione e modificando i pesi in accordo al gradiente calcolato per ciascun pattern, si converge al punto di minimo. Seguendo questo ragionamento, possiamo raggiungere il punto di ottimo, invece che nel modo più ovvio (cioè calcolandolo analiticamente) attraverso una serie di *spostamenti ripetuti*. La conseguente modifica dei pesi (che dev'essere di segno opposto rispetto a $\nabla_w F(w)$, dal momento che dobbiamo spostarci nella direzione discendente) è riportata nella stessa figura come Δw_k (si noti la somiglianza con quella del percettron, che quindi trova adesso una giustificazione). Adottando ad ogni passo di addestramento questa modifica dei pesi, ci spostiamo lungo la superficie di errore, ed è certo che dopo un certo numero di passi raggiungeremo il punto di minimo.

$$\nabla_w F(w) = \nabla[\lim_{N\to\infty} \frac{1}{N}\sum_{k=1}^{N}(y_k - y'_k)^2] =$$

$$= \lim_{N\to\infty} \frac{1}{N}\sum_{k=1}^{N} 2(y_k - y'_k)\nabla(y'_k) =$$

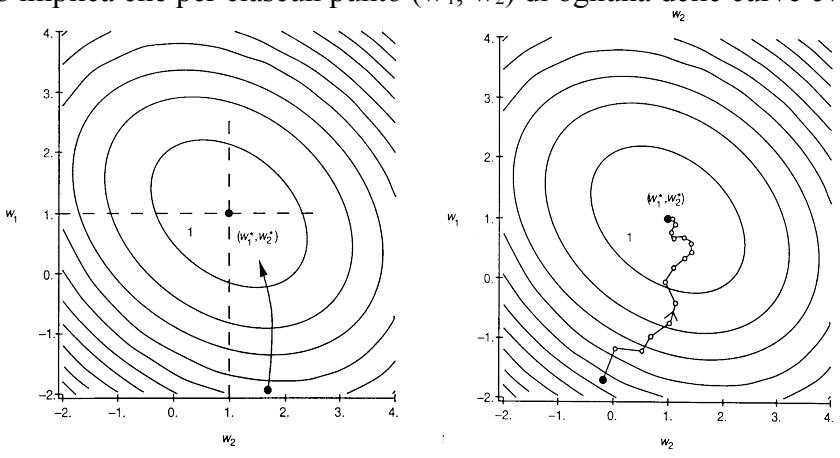
$$= \lim_{N\to\infty} \frac{1}{N}\sum_{k=1}^{N} 2(y_k - y'_k)(-x_k) =$$

$$= -2E[(y_k - y'_k)x_k]$$

$$\Delta w_k = \eta(y_k - y'_k)(x_k)$$

Ad ogni passo, nota per ogni campione l'uscita desiderata y_k e ottenute l'uscita di Adaline y'_k si calcola la differenza, la si moltiplica per x_k e per il coefficiente η di cui diremo fra breve, determinando la differenza dei pesi relativa a quel campione. Si ottiene in questo modo un gradiente 'locale' rispetto al campione e non mediato rispetto all'intero TS.

Riportiamo qui di seguito le 'curve di isoerrore'. Dato il paraboloide della superficie di errore, lo intersechiamo con una serie di piani paralleli. Siccome un piano parallelo denota un livello costante, ciò implica che per ciascun punto (w_1, w_2) di ognuna delle curve evidenzate il valore



dell'errore è lo stesso.

Calcolare in modo esatto una media statistica equivale a muoversi ortogonalmente rispetto alle curve di isoerrore (grafico a sinistra): ci stiamo cioè realmente muovendo nella direzione di massima pendenza. Il tragitto percorso è sicuramente il più breve possibile, tuttavia tale percorso è determinato solo una volta che si sia effettuato il calcolo dell'errore su tutto il TS. Anche se ciò fosse possibile, si avrebbe comunque un consumo notevole di tempo nella fase di computazione. Nel secondo caso invece (grafico di destra), effettuando la modifica dei pesi con il training by pattern nella filosofia di Widrow & Hoff, si riscontra uno spostamento 'a scatti' che localmente *non sempre si muove nella direzione corretta*, ma a regime converge[1] alla soluzione.

Veniamo al coefficiente η, detto *learning rate* (coefficiente di apprendimento): esso misura la dimensione del passo. La lunghezza di ciascun passo è proporzionale a η, sicché un learning rate molto piccolo causa una convergenza molto lenta, mentre se è troppo grande si corre il rischio di non riuscire a fermarsi mai sul punto di minimo (divergenza rispetto alla soluzione). Il valore del learning rate è in genere minore di 1 con valori tipici intorno a 0.5.[2]

[1] Nella pratica quindi ci si avvicina alla soluzione analitica a meno di un certo errore.

[2] Vengono eseguite alcune demo Matlab (basta digitarne il nome dalla riga di comando). Se ne riporta di seguito la descrizione in forma succinta.

DEMOP1 – percettrone a due ingressi, 4 vettori d'ingresso – processo converge – generalizzazione (corretta) per un quinto ingresso

DEMOP4 – classificazione in presenza di outlayer (vettore d'ingresso significativamente diverso da tutti gli altri, apprendimento più lento perché il vettore x è molto grande in norma)

DEMOP5 – leggera modifica alla regola di apprendimento di Rosenblatt, permette di avere una convergenza più rapida (basta normalizzare il vettore x)

DEMOP6 – fallimento in caso di vettori d'ingresso non linearmente separabili

DEMOLIN1 – Adaline: vettore d'ingresso monodimensionale – il problema è risolto analiticamente (non c'è fase di addestramento)

DEMOLIN2 – stesso esempio, legge di Widrow-Hoff con l'applicazione però del training by epoch invece del canonico training by pattern – è possibile scegliere il punto iniziale

DEMOLIN5 – problema non determinato: la superficie d'errore degenera, non c'è un unico punto di minimo. La soluzione trovata è quella cui corrisponde la norma più bassa

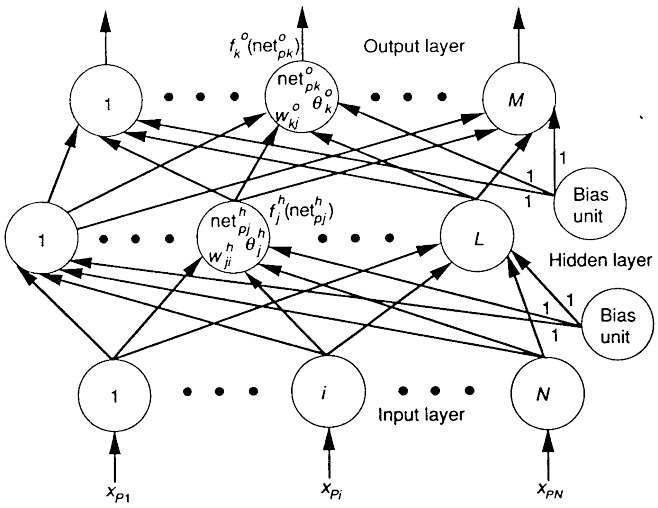
DEMOLIN7 – learning rate troppo alto, oscillazione divergente

DEMOLIN8 – Widrow & Hoff in senso stretto (training by pattern) – inseguimento di una funzione (verde scuro, poi rosso) – errore (verde chiaro) tende a zero

I modelli visti fin qui hanno ormai un valore quasi esclusivamente didattico. Presentiamo ora due architetture di maggior pregio: il Percettrone Multilivello e l'LVQ. Nel Percettrone Multilivello (MLP) i dati entrano in ingresso al primo layer, si propagano verso tutti i layer interni e da questi fluiscono verso l'ultimo. Il flusso di dati è unidirezionale e di tipo feed-forward: neuroni dello stesso livello non possono scambiarsi tra di loro dei dati e non c'è propagazione all'indietro. Si usa un algoritmo di apprendimento di tipo Performance Learning, simile a quello di Adaline. Abbiamo già accennato al fatto che il problema della rete MLP è che risulta difficile estendere la legge di apprendimento vista nel caso del singolo strato, dato che non conosciamo le uscite desiderate dei neuroni non appartenenti allo strato di uscita: ci torneremo nel corso di questa lezione.

L'LVQ differisce nella legge di apprendimento, che è di tipo Competitive Learning, e nel fatto che sono previste delle connessioni laterali.

La MLP si caratterizza per la presenza di almeno uno strato nascosto, che 'non vede' né gli ingressi x_{P1}… x_{PN} (essendo x_P il p-esimo esemplare dei Training Set) né le uscite della rete. Nelle nostre considerazioni faremo riferimento ad una MLP a 3 strati (un solo strato nascosto).



Il numero di ingressi di una RN è fissato dal problema: ad esempio per un approssimatore di una funzione a 3 variabili lo stato d'ingresso sarà composto da 3 neuroni. Gli N nodi dello stato iniziale non sono però dei neuroni in senso stretto, perché la loro ragione d'essere è di propagare i dati d'ingresso verso gli strati successivi, ma non effettuano alcuna computazione. Se non ci fosse lo strato nascosto rimarrebbe in pratica un solo strato di percettroni: *i Percettroni Multilivello presentano sempre almeno uno strato hidden*.

Le Bias Unit sono unità fittizie il cui scopo è di presentare a ciascun neurone un ingresso fisso pari a 1 (appare anche nei modelli del Percettrone e Adaline).

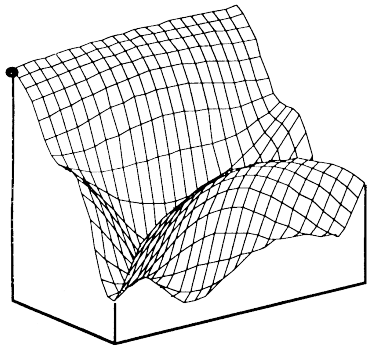
$$net^{h}_{pj} = \sum_{i=1}^{N} w^{h}_{ji} x_{pi} + \theta^{h}_{j} \qquad net^{o}_{pk} = \sum_{j=1}^{L} w^{o}_{kj} i_{pj} + \theta^{o}_{k}$$

$$i_{pj} = f^{h}_{j}(net^{h}_{pj}) \qquad o_{pk} = f^{o}_{k}(net^{o}_{pk})$$

Consideriamo le formule che seguono. Con net^{h}_{pj} si vuole indicare l'uscita del singolo neurone a meno della funzione di attivazione. Il generico neurone j effettua il prodotto scalare del vettore degli ingressi per il proprio vettore dei pesi, rappresentativi della memoria locale del percettrone; viene poi aggiunto il solito peso associato all'ingresso unitario. Gli apici **h** e **o** stanno rispettivamente per strato hidden e output, mentre **p** denota il p-esimo campione del TS. Alla combinazio-

ne lineare così calcolata viene applicata la funzione di attivazione del neurone, che nel caso del percettrone era un gradino, e nel caso di Adaline la funzione identità. Nel caso MLP non esiste un unico tipo di funzione di attivazione e non è detto che tutti gli strati presentino la stessa funzione di attivazione (benché in generale sia così).

Consideriamo ora l'addestramento di una MLP. Un primo aspetto notevole è che una MLP introduce delle non-linearità, a motivo della struttura delle funzioni di attivazione che considereremo fra breve. Questo è il vero vantaggio del modello, perché permette di risolvere problemi non linearmente separabili. Ciò d'altra parte determina una forma diversa per la superficie di errore, che non è più un paraboloide, ma un agglomerato spesso molto complesso di monti e valli. In questo modo il problema della ricerca del minimo si complica, soprattutto perché non si ha in generale un unico minimo, ma più minimi 'locali'.



L'idea di base è applicare alla MLP il principio ispiratore di Adaline. Definito l'errore E come nella formula sottostante (D = uscita desiderata, O = uscita prodotta dalla rete), modifichiamo ciascun peso di posto (i,j), collegante il neurone i-esimo di uno strato con il neurone j-esimo dello strato successivo, mediante un termine che sia indicativo del fatto che ci stiamo movendo nella direzione discendente del gradiente, e quindi sarà proporzionale secondo un coefficiente di tipo learning rate all'opposto della derivata parziale rispetto al peso considerato.

$$E = \frac{1}{2}\sum_j (D_j - O_j)^2$$

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$$

L'algoritmo di Back Propagation (BP). È costituito da due cicli. Il più esterno viene ripetuto finché non si raggiunge una condizione di terminazione la quale, dato che il modello rientra nella categoria del performance learning, consiste nella minimizzazione dell'errore E. L'addestramento cioè prosegue finché non si ritiene di aver raggiunto, a partire dai campioni presentati all'ingresso, un sufficiente livello di apprendimento.

L'apprendimento vero e proprio, ripetuto ad ogni occorrenza su tutti i campioni del TS, avviene nel ciclo interno. La particolare struttura di tale ciclo interno dà il nome all'algoritmo, che potrebbe essere tradotto come 'retro-propagazione'. Nella fase di **forward** il generico campione p viene posto in ingresso alla rete, ciascun neurone effettua i propri calcoli e lo strato di output presenta l'uscita relativa al p-esimo campione. Si effettua quindi una **valutazione dell'errore** come scarto quadratico (l'algoritmo è supervisionato: si sfrutta la conoscenza dell'uscita desiderata per ciascun campione e per ciascun neurone). Infine nella fase di **backward** si adopera il valore dell'errore appena calcolato per modificare di conseguenza i pesi secondo la formula vista per Δw_{ij}: in queste modifiche consiste in effetti l'apprendimento. L'algoritmo, di cui è stata matematicamente dimostrata la convergenza, segue un *training by pattern*, dato che la matrice dei pesi viene modificata per ogni singolo campione d'ingresso, ma potrebbe essere riscritto nella filosofia del *training by epoch*: sarebbe sufficiente spostare la fase di backward all'esterno del ciclo for, in modo che questa venga effettuata solo dopo aver calcolato l'errore quadratico sull'intero TS (si tratterà, a questo punto, un errore quadratico *medio* perché non più relativo ad un singolo esemplare). Il numero di epoche può essere più o meno elevato in ragione della cardinalità N_{tr} del TS: per 10.000 esemplari può essere sufficiente un centinaio di epoche, cosicché

```
PROCEDURE Back-Propagation learning ()
 BEGIN
      REPEAT
        E = 0;
        FOR  r := 1  TO  N_tr
         BEGIN
            forward();
            valutazione di E;
            backward();
         END;
      UNTIL (condizione di stop(E))
 END;
```

si abbia un milione di passi di addestramento. Il tempo di addestramento può essere quindi molto elevato; sono state proposte alcune variazioni sul tema di questo algoritmo finalizzate a migliorare l'efficienza.

$$E_p = \frac{1}{2}\sum_{k=1}^{M}\delta_{pk}^2 = \frac{1}{2}\sum_{k=1}^{M}(y_{pk} - o_{pk})^2$$

Riportiamo a destra le espressioni dell'errore relativo al p-simo campione e del gradiente, funzionali al calcolo della modifica al vettore dei pesi. o_{pk} e y_{pk} sono rispettivamente l'uscita effettiva e desiderata del neurone k. I passaggi sono ottenuti applicando il teorema di derivazione delle funzioni composte. Nel risultato finale appare la derivata della funzione di attivazione, per cui tale funzione dev'essere per ipotesi differenziabile.

$$\frac{\partial E}{\partial w_{kj}^o} = -(y_{pk} - o_{pk})\frac{\partial f_k^o}{\partial (net_{pk}^o)}\frac{\partial (net_{pk}^o)}{\partial w_{kj}^o}$$

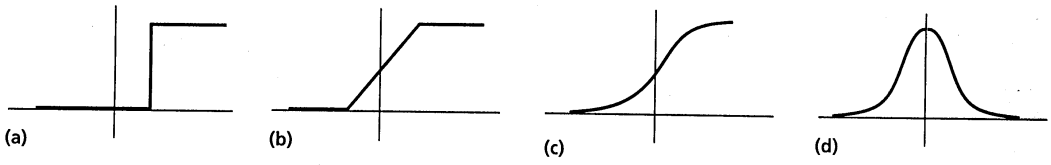
$$\frac{\partial (net_{pk}^o)}{\partial w_{kj}^o} = \left(\frac{\partial}{\partial w_{kj}^o}\sum_{j=1}^{L} w_{kj}^o i_{pj} + \theta_k^o\right) = i_{pj}$$

$$-\frac{\partial E}{\partial w_{kj}^o} = (y_{pk} - o_{pk}) f_k^{o\,\prime}(net_{pk}^o) i_{pj}$$

$$\Delta_p w_{kj}^o = \eta (y_{pk} - o_{pk}) f_k^{o\,\prime}(net_{pk}^o) i_{pj}$$

Possono essere scelte varie funzioni di attivazione: il grafico seguente ne illustra quattro tra le più comuni. La funzione a soglia (a) e lineare (b) presentano il problema di non essere derivabili in tutti i punti. Nei punti angolosi si potrà effettuare una derivazione per continuità. La (b) (in particolare, l'identità) è la funzione di attivazione di Adaline.

Un'altra è la funzione sigmoidale, per la quale è riportato in calce alla figura il calcolo della modifica ai pesi. Ricordiamo che 'net' sta per 'uscita netta', ovvero senza l'applicazione



(a) a gradino **(b) lineare a tratti** **(c) sigmoidale** **(d) gaussiana**

$$f_k^o(net_{jk}^o) = net_{jk}^o$$

$$f_k^o(net_{jk}^o) = \frac{1}{1 + e^{-net_{jk}^o}} \Rightarrow f_k^{o\,\prime}(net_{jk}^o) = f_k^o(1 - f_k^o) = o_{jk}(1 - o_{jk})$$

$$\Delta_p w_{kj}^o = \eta (y_{pk} - o_{pk}) o_{pk}(1 - o_{pk}) i_{pj}$$

della funzione di attivazione. La sigmoide, limitata tra 0 e 1, presenta la caratteristica di avere derivata (pendenza) massima intorno all'origine e derivata nulla. Ciò implica che quanto più l'uscita del neurone, che è sempre compresa tra questi due valori, si mantiene intorno al valore zero, tanto maggiormente l'algoritmo di apprendimento modificherà il vettore dei pesi. Infatti se tale uscita è in modulo molto grande darà luogo ad una derivata quasi nulla, dal momento che la sigmoide tende asintoticamente a due valori costanti (0 a sinistra e 1 a destra).

Diamo ora una risposta alla domanda lasciata da lungo tempo in sospeso: come modificare i pesi per lo strato nascosto, quello per il quale non conosciamo le uscite desiderate.

Il problema viene risolto molto semplicemente osservando che ciò che serve in ultima analisi è il gradiente dell'errore; non c'è una vera necessità di conoscere le uscite desiderate. Basta allora esplicitare per lo strato hidden la dipendenza dai pesi da parte dell'errore, e applicare la regola di derivazione delle funzioni composte. L'uscita o_{pk} del neurone di output dipende dal i_{pj}, che è a sua volta dipendente da w_{ji}. i_{pj} è l'uscita prodotta del j-simo neurone dello strato hidden, che dipende evidentemente non solo dagli ingressi ma anche dai pesi del neurone stesso (ovvero i pesi ingresso-hidden).

Si noti come sono state espresse le derivate parziali. Due di esse sono semplicemente il peso w_{kj}^{o} e l'ingresso x_{pi}. Le altre due sono derivate della funzione di attivazione, che dunque anche per lo strato hidden dev'essere differenziabile.

$$E^{h} = \frac{1}{I}\sum^{K}(\lambda^{hK} - o^{hK})_{3} = \frac{1}{2}\sum_{k}(y_{pk} - f_{k}^{o}(\sum_{j=1}^{L} w_{kj}^{o} i_{pj} + \theta_{k}^{o}))^{2}$$

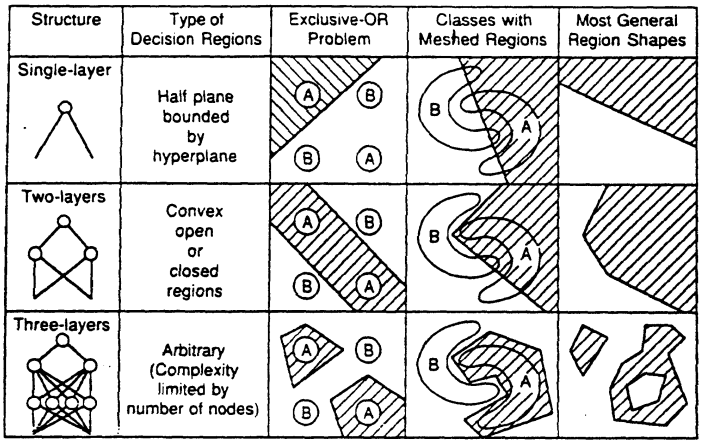
$$\frac{\partial E}{\partial w_{ji}^{h}} = \frac{1}{2}\sum_{k}\frac{\partial}{\partial w_{ji}^{h}}(y_{pk} - o_{pk})^{2} =$$

$$= -\sum_{k}(y_{pk} - o_{pk})\frac{\partial o_{pk}}{\partial (net_{pk}^{o})}\frac{\partial (net_{pk}^{o})}{\partial i_{pj}}\frac{\partial i_{pj}}{\partial (net_{pj}^{h})}\frac{\partial (net_{pj}^{h})}{\partial w_{ji}^{h}}$$

$$\frac{\partial E}{\partial w_{ji}^{h}} = -\sum_{k}(y_{pk} - o_{pk}) f_{k}^{o\,\prime}(net_{pk}^{o}) w_{kj}^{o} f_{j}^{h\,\prime}(net_{pj}^{h}) x_{pi}$$

$$\Delta_{p} w_{kj}^{h} = \eta f_{j}^{h\,\prime}(net_{pj}^{h}) x_{pi} \sum_{k}(y_{pk} - o_{pk}) f_{k}^{o\,\prime}(net_{pk}^{o}) w_{kj}^{o}$$

MLP: funzionamento da classificatore. Nel classificatore MLP si usa per le uscite un 'codice decodificato': si mettono nello strato di uscita tanti neuroni quante sono le classi. Ad esempio, se abbiamo tra classi A, B e C e si presenta un ingresso A, ci aspettiamo che l'uscita varrà 1 per il neurone di uscita relativo alla classe A e 0 per gli altri due. In pratica, l'uscita della rete non sarà in generale binaria, ma composta di numeri reali, ad esempio i neuroni dell'ultimo strato forniscono le uscite A=0.8, B=0.11, C=0.09. Si adotta allora l'ovvia strategia del tipo 'winner takes all' (WTA): 'vince' la classe associata al neurone che ha l'uscita più alta. Il campione in ingresso viene attribuito quindi alla classe A. Naturalmente tale strategia può dar luogo a delle incertezze. Infatti due delle uscite potrebbero essere troppo vicine fra di loro perché si possa ritenere vincente una delle due.

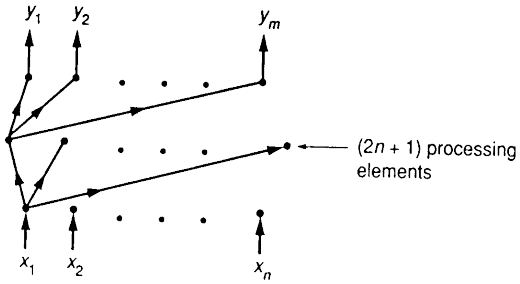


Il disegno che segue illustra il comportamento di un percettrone in vari contesti. Facciamo l'ipotesi di dover discriminare fra due classi soltanto. In presenza di un single layer le regioni di decisione sono dei semipiani limitati da un iperpiano. Per il problema della xor e quello subito dopo, in cui figurano regioni del piano posizionate in modo da rendere necessaria una separazione piuttosto articolata, la struttura a single-layer fallisce senz'altro. La struttura a due livelli è in grado di generare regioni di decisioni convesse, ma aperte. I due percettroni dello strato hidden dividono il piano in semipiani mediante delle rette; lo strato di uscita 'compone' le regioni di decisione. Se lo strato hidden conta tre nodi, otterremo una regione aperta delimitata da tre rette come nell'esempio più a destra. La struttura riesce così a risolvere il problema della xor, ma, con due nodi soltanto, non il secondo problema. Nel caso di tre strati (due nascosti) la forma delle regioni di decisioni può essere qualunque: possono essere anche 'forate'. Tre strati sono quindi teoricamente sufficienti a risolvere qualsiasi problema. Tuttavia all'aumentare del numero di strati la convergenza diviene molto più lenta, per cui generalmente si opta per una struttura a due strati con un numero adeguato di neuroni.

Fra i problemi presentati dalla reti a percettroni vi è la presenza di minimi locali, problema che si cerca di aggirare ricorrendo a opportune variazioni nella back propagation. Conseguenze negative sono legate anche all'overtraining: un eccessivo addestramento della rete. La rete si specializza troppo, e l'errore tende addirittura ad aumentare col numero di epoche. Citiamo anche il problema del flat spots: una errata inizializzazione dei pesi (es. positivi ed elevati) può portare in una zona dell'asse delle ascisse nel piano della funzione di attivazione in cui la derivata è praticamente nulla. In particolare, i pesi devono essere sia positivi che negativi, ed il loro valore dev'essere

relativamente piccolo rispetto al numero N di neuroni: generalmente il loro valore è scelto come compreso tra $\pm 1/\sqrt{N}$.

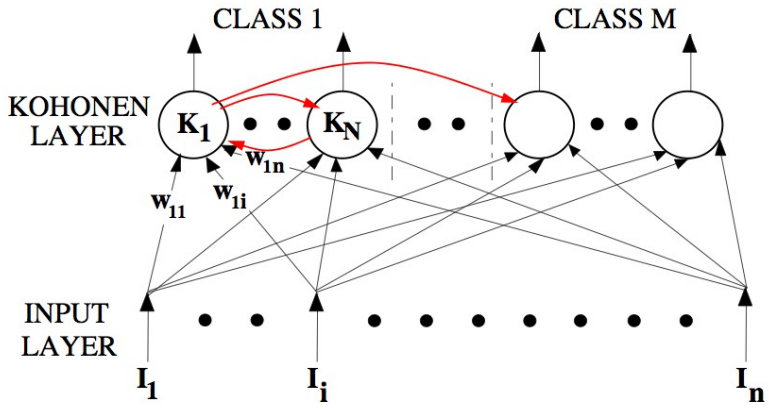
Il teorema di Kolmogorov. Consideriamo una qualunque funzione continua ad n ingressi ed m uscite, che assume valori tra 0 e 1. Essa può essere esattamente implementata da una RN feed-forward a 3 strati, con (2n+1) elementi nello strato nascosto e m nello strato di uscita. La funzione di trasferimento del secondo strato è semilineare, ovvero simile ad una somma pesata lineare. Quella del terzo strato è invece altamente non lineare. Il teorema non specifica però la forma di quest'ultima funzione di trasferimento, dicendo solo che dipende da certi parametri, né indica come calcolare i pesi. La sua utilità pratica quindi è modesta.



L'ARCHITETTURA LVQ. Questa rete presenta due strati, uno di input e uno di computazione/uscita, detto strato di Kohonen. Funziona da **classificatore**: attribuisce un elemento a una fra M possibili classi. Allo scopo, in fase di definizione dell'architettura i neuroni sono etichettati in modo da appartenere alle varie classi. A differenza del multilayer perceptron, si hanno in generale più neuroni per ciascuna classe. Il numero di neuroni associati alle classi può essere differente per le varie classi.

Alla fine del processo di addestramento, ciascun neurone rappresenterà un prototipo dei campioni della classe alla quale esso è ascritto. In altre parole, ciascun neurone modellerà i soli campioni del TS che appartengono alla classe a cui è stato preventivamente associato. Nella figura che segue, N neuroni sono stati assegnati alla classe 1 e ne dovranno modellare i campioni.

Questo è un punto molto importante. Nel modello MLP non c'è modo di dire, osservando la configurazione della rete, come la conoscenza è distribuita in tutti i suoi punti. Di conseguenza una MLP non è neanche in grado di giustificare le risposte che fornisce. Nel caso in esame invece i neuroni danno più informazione. È più facile effettuare un 'post-processing' del risultato della rete, spiegando il modo in cui è stato ottenuto.



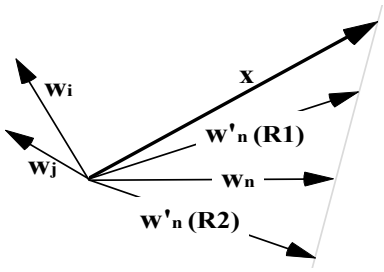
L'apprendimento è di tipo *competitivo*. Si noti che ciascun ingresso è connesso a tutti i neuroni. I pesi hanno un significato diverso rispetto al caso MLP: non viene calcolata una somma pesata, ma ciascun neurone determina la distanza tra il vettore d'ingresso ed il proprio vettore dei pesi i due vettori hanno la stessa dimensione). Il vettore dei pesi attribuito a ciascun neurone rappresenta appunto il prototipo dei campioni del TS cui si accennava prima. Quando un vettore si presenta in ingresso alla RN, si scatena la **competizione**: si stabilisce quale dei neuroni si trova a distanza minima da esso, cioè quello il cui vettore dei pesi, rappresentativo di una certa classe, è più vicino all'ingresso. La metrica usata per computare tale distanza è quella euclidea (radice quadrata della

somma dei quadrati delle differenze). Il neurone vincitore è l'unico al quale è concesso di modificare i propri pesi.

In fase operativa viene presentato un ingresso e ciascun neurone calcola la distanza rispetto ai propri pesi. L'uscita dei neuroni varrà 1 per quello il cui vettore dei pesi è maggiormente vicino a quello degli ingressi, 0 per tutti gli altri. Ovviamente per conoscere l'identità del neurone vincitore è necessario che ogni neurone scambi informazioni con i tutti gli altri del proprio strato. Ecco perché la rete è a connessioni laterali. Queste ultime sono usate dai neuroni per inviare ai propri compagni di strato la distanza dall'ingresso. Naturalmente questo è vero solo in senso formale: all'atto pratico sarà sufficiente utilizzare un algoritmo che calcoli il minimo in un vettore.

Abbiamo quindi introdotto il paradigma LVQ. Soltanto un neurone può cambiare i propri pesi in fase di addestramento: quel neurone che ha minima distanza dall'ingresso. L'utilizzo tipico è di classificatore ad M classi; N neuroni sono attribuiti a ciascuna classe. Ciascun neurone rappresenta un prototipo per gli elementi della classe che rappresenta.

Consideriamo l'algoritmo LVQ1. Con w_i, w_j e w_n abbiamo rappresentato i vettori dei pesi associati ai neuroni dello strato di Kohonen: nel nostro esempio supponiamo che siano 3. Immaginiamo che ogni ingresso x abbia due dimensioni (due componenti). Ciascun neurone calcola la distanza euclidea fra il proprio vettore dei pesi e l'ingresso x. Il neurone w_n che ha la distanza minima d_i 'vince' la competizione. Si verifica quindi se l'attribuzione dell'ingresso alla classe del neurone vincitore è corretta. Tale verifica è possibile, perché essendo l'apprendimento supervisionato si conosce la classe di appartenenza dell'ingresso. In caso affermativo, il neurone viene 'premiato' avvicinandolo alla classe x (**regola R1**). Altrimenti w_n viene penalizzato allontanandolo dalla classe in questione (il riconoscimento è errato: **regola R2**). In tal modo, quando x si ripresenterà in ingresso è più probabile che il neurone 'giusto' (quello della classe x) sia abbinato ad esso.



R1: $\Delta w_n = \alpha(x-w_n)$

R2: $\Delta w_n = -\gamma(x-w_n)$

```
PROCEDURE  Competitive Learning( )
 BEGIN
   FOR  i := 1 TO N*M   DO
                        di = distanza(x,wi) ;
   n = minimo(di) ;
    IF classe[n] = classe[x]
           THEN aggiorna con la regola R1
           ELSE  aggiorna con la regola R2
 END
```

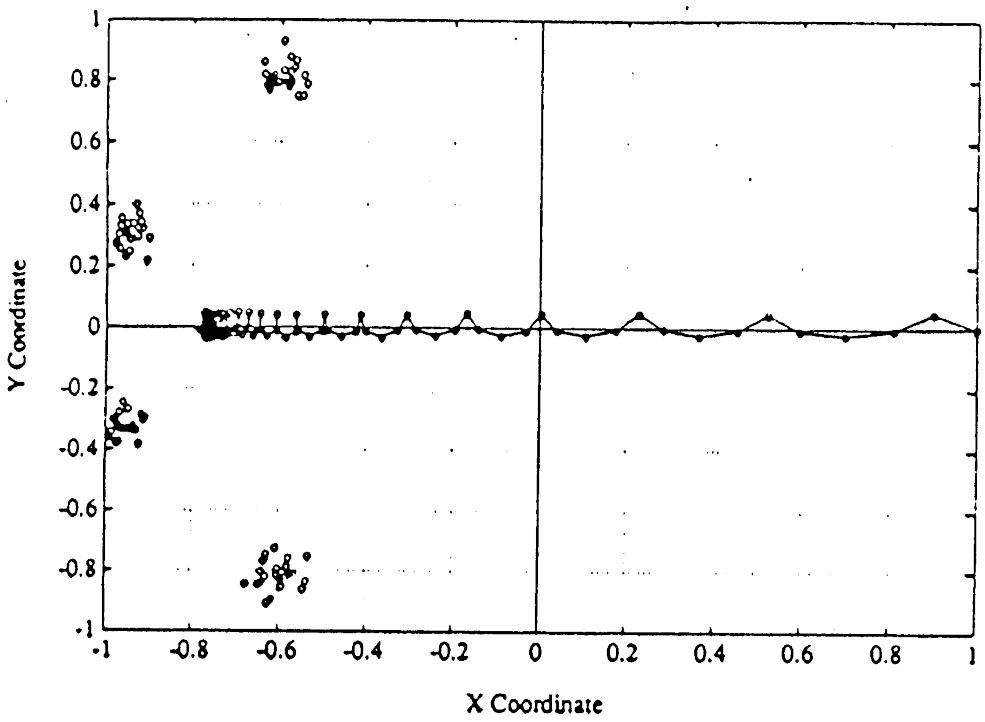
Dal punto di vista geometrico, tutto ciò equivale a sommare o sottrarre al vettore w_n una frazione del vettore $(x-w_n)$ che congiunge w_n a x, ottenendo il nuovo vettore w'_n (vedi il grafico con i vettori). I coefficienti α e γ devono essere tarati in maniera opportuna; sono infatti i responsabili della convergenza dell'algoritmo. In particolare, essi non devono essere costanti ma variabili nel tempo. Si fa in modo che essi assumano valore massimo 1 all'inizio per poi decrescere verso lo zero linearmente con il numero di epoche ($\alpha = \alpha_0(1-T/T_{MAX})$, essendo T_{MAX} il numero massimo di epoche e T l'epoca corrente).

L'algoritmo, che è supervisionato nella versione base, può essere reso non supervisionato per problemi di *clustering*: si lascia che i prototipi evolgano liberamente, occupando in corso di apprendimento una determinata posizione nello spazio delle caratteristiche per divenire rappresentativi dei campioni. In questa variante, i neuroni non sono preassegnati alle classi, e il neurone vincitore viene sistematicamente avvicinato alla classe di appartenenza dell'ingresso: scompaiono le righe di codice che iniziano per **IF** ed **ELSE**.

La figura che segue fa riferimento proprio al caso non supervisionato. Si hanno 4 raggruppamenti o *cluster* di campioni (i punti). Supponiamo di avere uno strato di Kohonen con 4

neuroni. Desideriamo pertanto che alla fine del ciclo di addestramento i 4 neuroni vadano a posizionarsi ciascuno in uno dei 4 cluster assegnati. Ipotizziamo che tutti i neuroni abbiano dei pesi tali da essere concentrati in partenza nel punto (1,0).

Si presenta il primo campione del TS. Poiché tutti i neuroni hanno il vettore rappresentativo nello stesso punto (1,0), è evidente che avranno tutti e 4 la medesima distanza dal campione in questione. Se ne sceglie uno a caso, a cui viene dato l'agio di modificare il proprio vettore dei pesi. Il neurone vincitore si avvicina al campione d'ingresso. Quando si presenta un secondo campione, a qualunque cluster esso appartenga (anche se diverso dal primo), il neurone che ha vinto per estrazione a sorte la prima competizione vincerà, questa volta legittimamente, anche la seconda, avvicinandosi (mediante una modifica al vettore dei pesi) al campione d'ingresso. Il neurone si avvici-



na sempre di più alla zona dello spazio dove sono localizzati i cluster; gli altri tre rimangono 'fermi al palo'. Ciò è del tutto indipendente dall'ordine in cui si presentano i campioni d'ingresso.

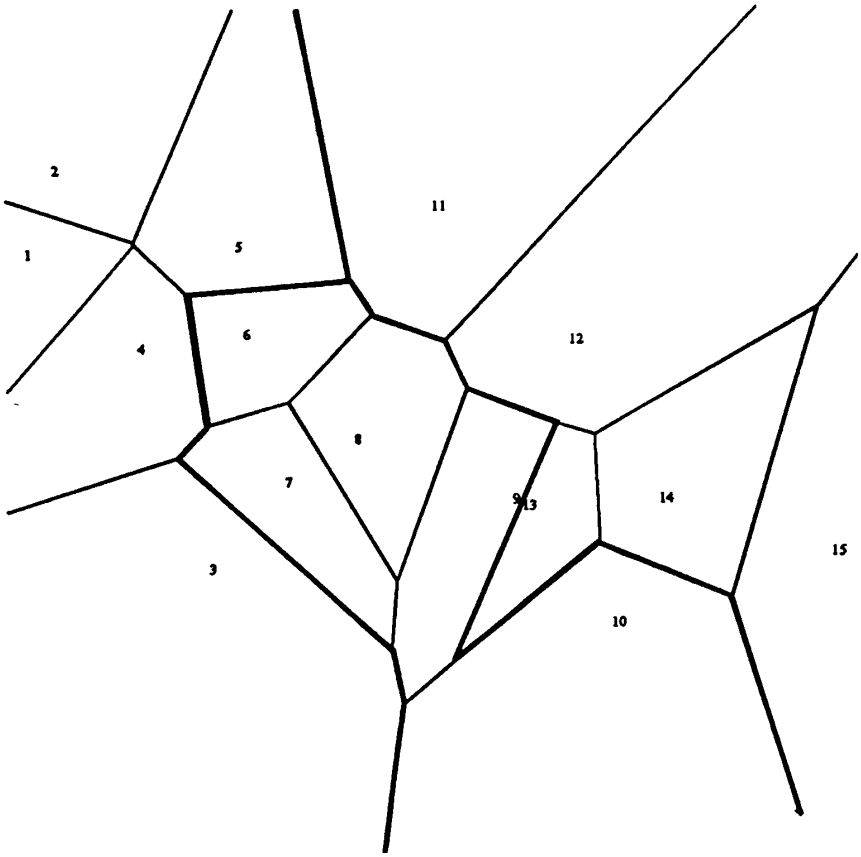
Ciò dà luogo ad un problema abbastanza fastidioso: quello che va sotto il nome di **sottoutilizzo** dei neuroni. Il neurone vincitore si avvicina con traiettoria oscillante ai cluster. Non si ha vera e propria convergenza, e per di più tutti e 4 i cluster vengono ad essere rappresentati dallo stesso neurone, che si mantiene in qualche modo equidistante da essi: gli altri tre non entrano mai in gioco. Tale problema si riscontra, sia pure con effetti meno critici, anche nel caso supervisionato.

Varie proposte sono state avanzate per risolvere il problema: parleremo di due di esse in particolare, l'algoritmo cosiddetto 'della coscienza' e l'algoritmo FSCL. Essi sono accomunati dalla scelta di ricorrere ad una definizione non costante di distanza (quella che viene usata per la competizione); i neuroni che 'vincono troppo spesso' vengono penalizzati rispetto agli altri, in modo che tutti abbiano le stesse opportunità di vittoria.

Per FSCL (Frequency Sensitive Competitive Learning) la modifica nel calcolo della distanza consiste nel moltiplicare la distanza euclidea per la frequenza relativa di vittorie del neurone: $\mathbf{d'_i = d_i * F_i}$. Questo algoritmo modifica non uno ma due neuroni per volta: vengono considerati i due neuroni più vicini all'ingresso; il primo viene ulteriormente avvicinato, il secondo ne viene invece allontanato. FSCL costituisce un approccio generalmente efficace, ma è possibile dimostrare anche il suo fallimento in certi casi.

In C^2L (Conscience Learning) la distanza viene diminuita attraverso il peso (bias) bi, secondo le formule $\mathbf{d''_i = d_i - b_i}$ e $\mathbf{b_i = c*(1/N - f_i)}$. 'c' è un valore intero, N il numero dei neuroni e f_i

la frequenza relativa di vittoria. Si osservi che in caso di equiprobabilità di vittoria (è la situazione ottimale), $f_i = 1/N$ e quindi $b_i = 0$, cioè la distanza del neurone è esattamente quella euclidea d_i. Se un neurone vince troppo spesso, la sua frequenza di vittoria diviene maggiore di 1/N: finiamo con l'aggiungere qualcosa a d''_i, invece di diminuire tale quantità: il neurone che vince troppo viene penalizzato.

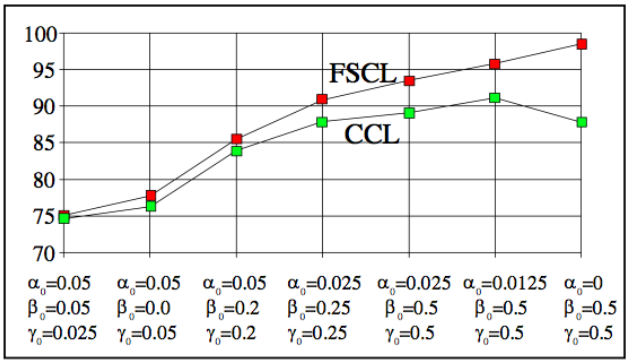


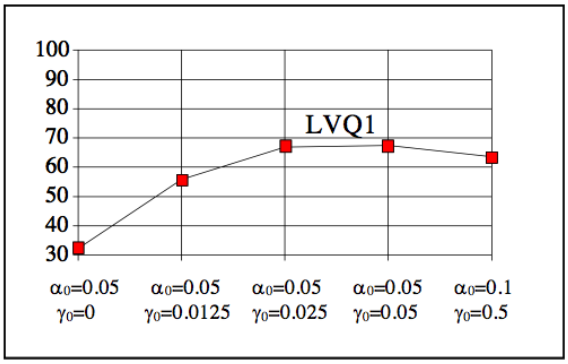
La rete LVQ realizza la cosiddetta 'Tassellazione di Voronoi' (vedi figura sovrastante). I numeri piccoli rappresentano ciascuno un punto dello spazio in cui vengono a trovarsi i vari neuroni: il neurone numero 12 si trova nel punto occupato dal numero. Per capire qual è la regione di decisione occupata dal neurone 12 basta considerare tutti i punti aventi distanza da tale neurone che è minore della distanza di altri neuroni. Si noti ad esempio che la linea che separa i neuroni 11 e 12 è il luogo dei punti equidistanti da tali neuroni (è facile rendersene conto). L'insieme delle linee così costruite (perpendicolari alle congiungenti di ogni coppia di neuroni nei punti medi) delimita le regioni di decisione. Posizionando opportunamente i neuroni, possiamo ottenere configurazioni di regioni di decisione comunque complesse, incluse quelle del multilayer perceptron a 3 strati.

I grafici che seguono fanno vedere come l'algoritmo LVQ1, per particolari sottoinsiemi di dati, raggiunga prestazioni poco lusinghiere: comunque si faccia variare il learning rate, esso non supera il 70% del riconoscimento (con un valore di n molto elevato). Con lo stesso TS, FSCL e C^2L raggiungono prestazioni migliori (in particolare il primo, che sfiora il 100% di corretta classificazione). Ovviamente utilizzando il Test Set anziche il TS (Training Set) si hanno prestazioni inferiori di un buon 5%[3].

[3] Vengono eseguito le seguenti DEMO Matlab :

GENERALIZATION – la rete deve riconoscere una funzione assegnata per punti. È possibile aumentare il grado di difficoltà del problema ed il numero di neuroni dello strato nascosto (fino a 9). Con pochi neuroni (e problema di opportuna complessità), la risposta della rete è di qualità scadente perché lo è l'apprendimento (*underfitting*: pochi gradi di libertà). Si noti che si può avere una generalizzazione scadente anche con troppi neuroni e problemi semplici (es.9 neuroni, difficoltà 7 o 8). Questo fenomeno è detto di *overfitting*: un numero di gradi di libertà troppo elevato (troppi neuroni ovvero 'troppe rette' possono essere posizionate nello spazio, causando una eccessiva specializzazione).



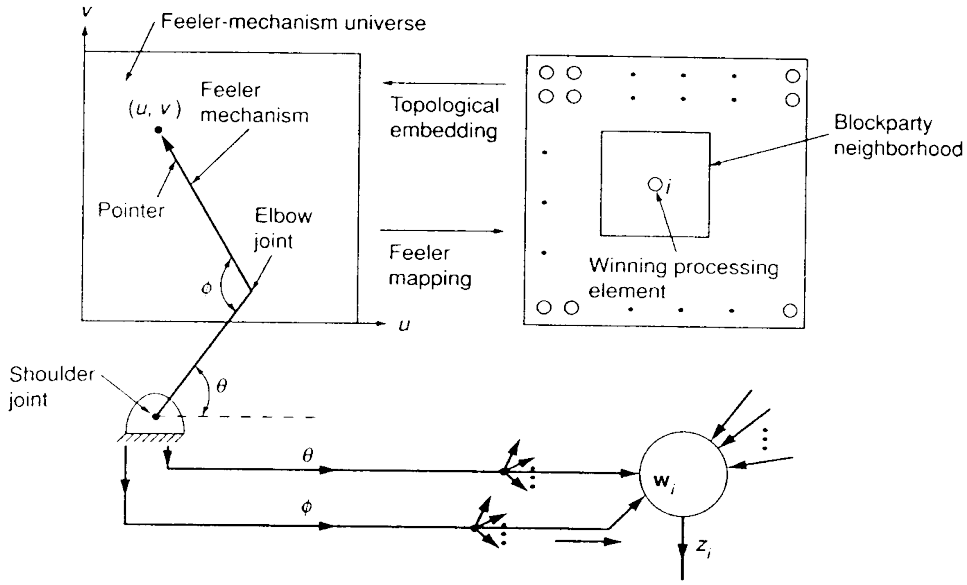


LE MAPPE AUTOORGANIZZANTI DI KOHONEN (SOM). Questo strumento presenta due differenze sostanziali rispetto al meccanismo LVQ. Innanzitutto, l'apprendimento è sempre non supervisionato. Inoltre, dal punto di vista topologico, i neuroni sono disposti *a griglia*. La posizione di ciascun neurone nella griglia è definita in un modo simile a quanto avviene nelle matrici: il neurone (1,2) sarà quello situato nella riga 1, colonna 2 della griglia. La topologia è descritta attraverso *connessioni* che hanno però un significato diverso rispetto ai casi visti fin qui (non c'è difatti un *peso* associato a ciascuna connessione). La rete di Kohonen viene sollecitata con certi punti; ciascuno di essi, in uno spazio (u,v), rappresenta il punto terminale di un 'braccio' a due giunti. Alla rete pervengono quali ingressi non le coordinate (u,v), ma i due angoli θ e φ che ne determinano la posizione (v.figura). È la rete stessa a realizzare la giusta corrispondenza topologica tra i due spazi. Ad esempio il neurone (1,1) si auto-organizza, divenendo rappresentativo dei punti dello spazio (θ, φ) corrispondenti a punti dello spazio (u,v) a loro volta corrispondenti all'angolo in alto a sinistra della griglia.

STEEPEST DESCENT BACK PROPAGATION – abbiamo 4 pesi e 3 pesi di bias, quindi 7 dimensioni; possiamo ovviamente rappresentarne al massimo 2 per volta (nel grafico della funzione di errore). La forma non è più quella di un paraboloide. La crocetta rossa segnala il minimo assoluto. Con il mouse è possibile scegliere le condizioni iniziali e vedere in quale direzione muove l'algoritmo di addestramento (se cioè converge o no verso il minimo assoluto). È usata la regola della direzione discendente del gradiente, il quale come si vede non sempre funziona.

COMPETITIVE LEARNING – versione non supervisionata. Gli 8 neuroni si posizionano sui gruppi di campioni, scelti con distribuzione casuale. In base alla configurazione di questi e all'ordine in cui vengono selezionati si può avere un sottoutilizzo.

LEARNING VECTOR QUANTIZATION – supervisionato. 2 classi (4 + azzurri e 6 + rossi). I neuroni si posizionano in modo corretto.



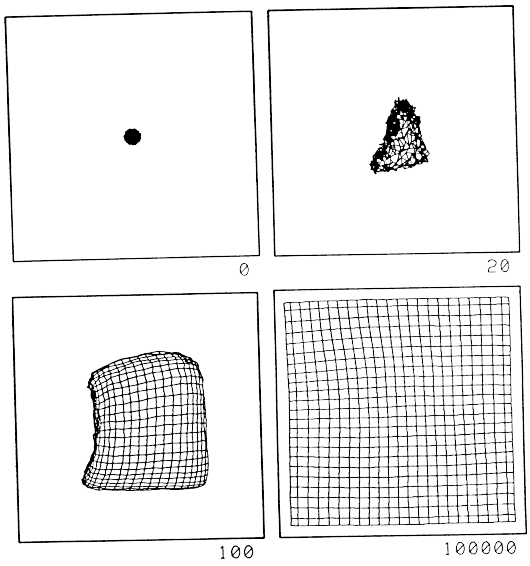
Nelle reti LVQ un solo neurone poteva muoversi in occasione di ciascun ciclo di addestramento. Come si è visto, ciò può portare come conseguenza il sottoutilizzo dei neuroni. Nelle SOM il problema del sottoutilizzo viene aggirato in un modo del tutto peculiare. Consideriamo la

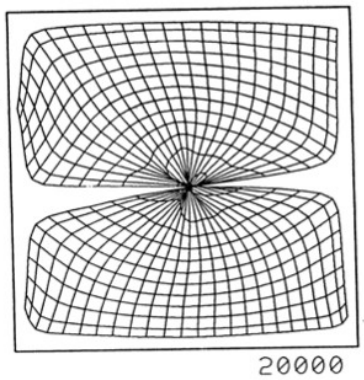
$$\mathbf{w}_i^{new} = \alpha_i(\mathbf{z},\mathbf{t})(\theta,\varphi) + [\mathbf{1} - \alpha_i(\mathbf{z},\mathbf{t})]\mathbf{w}_i^{old}$$

legge di apprendimento:

Ricordiamo che (θ, φ) è l'ingresso x. Se trascuriamo l'elemento (z,t), abbiamo una legge molto simile a quella di LVQ. Una prima differenza sta nel fatto che in linea di principio *i pesi di tutti i neuroni possono essere modificati*. Tuttavia il coefficiente α_i varia, oltre che col tempo t (decresce nel tempo), anche col parametro **z**, a sua volta dipendente dalla **distanza del neurone i-esimo dal neurone vincitore**. La distanza in questione non è da intendersi in senso euclideo, ma misurata sulla griglia di cui si diceva. Ad esempio i neuroni (1,2) e (2,2) sono a distanza 1. Quindi, ciascun neurone ha 4 neuroni a distanza 1 da sé stesso. In particolare, α_i *diminuisce* quanto più ci si allontana sulla griglia. Quindi insieme al neurone vincitore, sono 'coinvolti' nella vittoria quelli che si trovano nei suoi paraggi; anch'essi cioè sono messi in condizione di modificare significativamente i propri pesi; per tale motivo $_i$ viene detto fattore di *neighbour* (=vicinanza). L'intorno dei neuroni che risultano avvantaggiati del neurone vincitore può dunque variare nello spazio (possono essere coinvolti tutti i neuroni a distanza massima 1, o a distanza massima 2 etc.) e nel tempo.

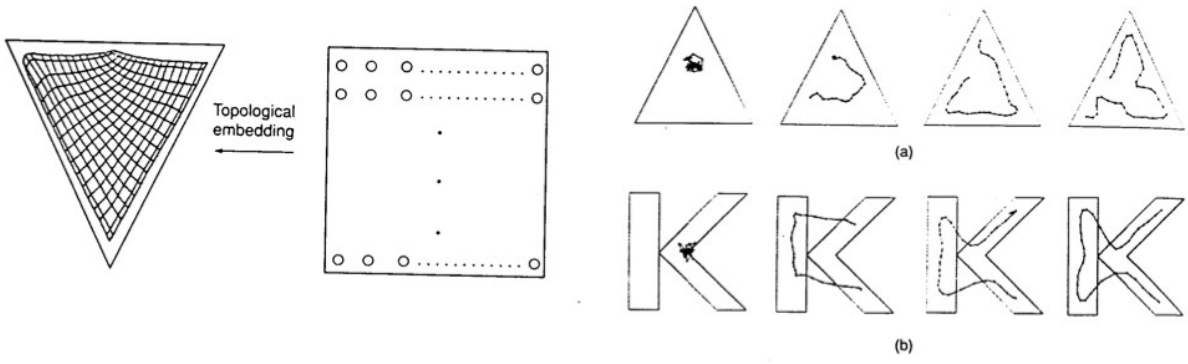
Si notino i grafici a destra. Inizialmente, tutti i neuroni di Kohonen sono concentrati nell'origine (ovvero lo sono i loro vettori dei pesi). Quando presentiamo alla rete ingressi (θ, φ) corrispondenti a punti (u,v) del piano, 'vincerà' il neurone il cui vettore dei pesi ha la distanza più piccola dall'ingresso. Tale neurone prende a muoversi (in un piano equivalente a (u,v)) portandosi appresso un certo numero di altri neuroni. All'aumentare dei cicli la rete di neuroni tende ad espandersi; dopo 100.000 cicli i neuroni avranno occupato tutto lo spazio disponibile. Una configurazione 'regolare' come quella del grafico in basso a destra è possibile tuttavia solo se anche la configurazione iniziale denotava una certa corrispondenza topologica, ad esempio se il neurone (1,1) si trovava nell'angolo nord-ovest dell'agglomerato iniziale, il neurone (1,N) in quello nord-est e via dicendo.



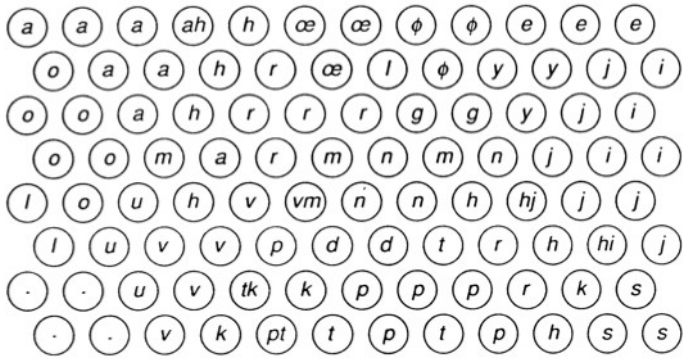


In caso contrario la rete finale può risultare 'intrecciata' come quella del disegno a sinistra (*twisted mesh*).

La SOM può essere stimolata con punti prelevati da un spazio bidimensionale diverso, ad esempio di tipo triangolare. Possiamo anche avere SOM lineari (e non griglie bidimensionali). In ogni caso, i neuroni si posizionano in modo da occupare tutto lo spazio disponibile, comportandosi alla stregua di un gas che tende ad assumere la forma del recipiente che lo contiene.

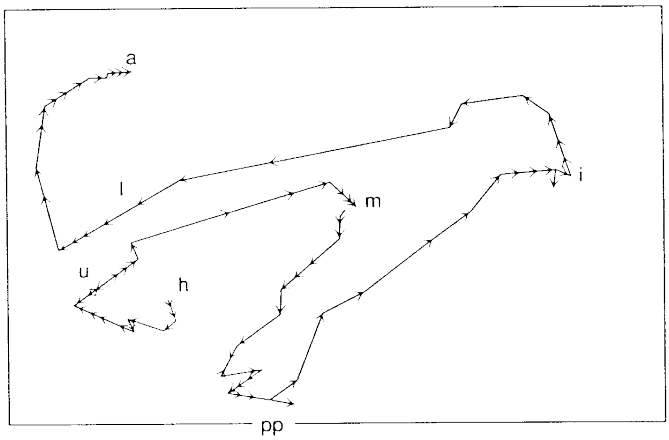


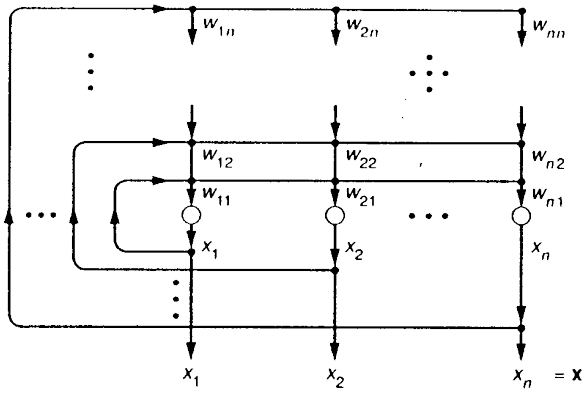
Interessante applicazione delle SOM di Kohonen è lo scrittore fonetico realizzato dallo stesso Kohonen (1988). Lo strumento era un rudimentale riconoscitore vocale capace di far apparire sullo schermo le lettere di una parola pronunciata da un utente umano. Le parole venivano opportunamente campionate (9.83 ms), e a ciascuna parola si associava un vettore a 15 componenti, ciascuna dei quali corrispondente ad una componente frequenziale della parola stessa. La rete veniva quindi sollecitata presentando in ingresso i risultati del campionamento delle parole.



Si noti la 'mappa fonotòpica' a sinistra, in cui a ciascun neurone è associata una lettera. Punto di partenza è realizzare il fatto che le SOM, pur non essendo supervisionate, possono essere usate come classificatori. Questo lo si può fare, dopo un primo training non supervisionato, sottoponendo nuovamente il TS alla mappa auto-organizzante. In questo secondo caso il training sarà però 'labellato'. Supponiamo di conoscere la classe di appartenenza di ciascun campione: sappiamo ad esempio che un certo numero di esemplari si riferiscono alla lettera 'a'. Per ciascun neurone andiamo quindi a vedere per quali campioni del TS è risultato essere il neurone vincitore. Attribuiremo quel neurone alla classe per la quale si siano verificati il maggior numero di riconoscimenti. Ad esempio se un neurone ha vinto 12 competizioni, e in 10 occasioni su 12 è stato associato alla classe 'lettera a', etichetteremo quel neurone come appartenente alla 'classe a'. Se il neurone vince 15 competizioni, e risulta essere 8 volte vicino alla 'classe a', e le altre 7 alla 'classe h', esiste evidentemente una incertezza: quel neurone viene attribuito ad una classe 'ah', per cui, se è lui a vincere in fase di esecuzione, il suono potrebbe essere tanto quello di una 'a' quanto quello di una 'h'.

A destra è indicato l'esempio della pronuncia della parola finlandese *humppila*. La parola viene campionata a 9.83 ms; ogni singolo campione viene processato dalla rete attraverso la sequenza di attivazioni dei neuroni indicata a destra. Con questo strumento si riusciva ad ottenere una percentuale di riconoscimento corretto del 92-97%.





LA RETE DI HOPFIELD. È un esempio *rete ricorrente*, per il quale non studieremo un algoritmo di addestramento. Ci sono differenze significative rispetto ai precedenti modelli. Ogni neurone è collegato a tutti gli altri attraverso dei pesi. Lo stesso vettore x è contemporaneamente ingresso, stato e uscita della rete. L'utilizzo tipico della rete di Hofield, come si è detto, è quello di memoria associativa: presentiamo un ingresso, e la rete ricerca lo *stato stabile* più vicino a tale ingresso. In fase di apprendimento la rete memorizza un certo numero di informazioni, detti stati stabili (o configurazioni), rappresentati da vettori x_i. In modalità operativa la rete, ricevuto un vettore iniziale, ricalcola – modificandolo – il proprio stato sulla base dei pesi colleganti i vari neuroni. La funzione di attivazione che permette il ricalcolo dello stato è indicata di seguito.

$$x_i^{new} = \begin{cases} 1 & \text{if } \sum_j w_{ij} x_j^{old} > T_i \\ x_i^{old} & \text{if } \sum_j w_{ij} x_j^{old} = T_i \\ -1 & \text{if } \sum_j w_{ij} x_j^{old} < T_i \end{cases}$$

I neuroni evolvono quindi, modificando il proprio stato, fino al punto che la rete raggiunge una configurazione stabile (memorizzata) simile a quella che è stata presentata in ingresso. Durante ogni ciclo, ciascun neurone x_i verifica se è il caso di modificare il proprio stato. Lo stato memorizzato, come si vede, è binario (può valere soltanto +1 o -1), quindi gli stati stabili che possono essere raggiunti sono vettori di +1 o –1. Ogni neurone calcola la somma pesata dell'ingresso per il vettore dei pesi delle connessioni che lo congiungono a tutti gli altri neuroni della rete (si noti che ogni neurone è connesso anche a sé stesso). Se il risultato è superiore ad una soglia, il nuovo stato dell' i-esimo neurone è 1; se è inferiore, è –1; se uguale alla soglia, non si ha modifica dello stato.

Si può dimostrare che il processo converge. Alla rete viene associata una funzione di energia; il suo valore diminuisce (o rimane inalterata) in corrispondenza di ogni modifica dei pesi. Quindi dopo un certo numero di iterazioni si raggiunge senz'altro il minimo di tale funzione (quando H=0 per tutti i neuroni, siamo pervenuti ad uno stato stabile).

$$H(x) = -\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} x_i x_j + 2\sum_{i=1}^{n} T_i x_i$$

$$\Delta H = H(x^{new}) - H(x^{old}) =$$

$$-\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} x_i^{new} x_j^{new} + 2\sum_{i=1}^{n} T_i x_i^{new} + \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} x_i^{old} x_i^{old} - 2\sum_{i=1}^{n} T_i x_i^{old} =$$

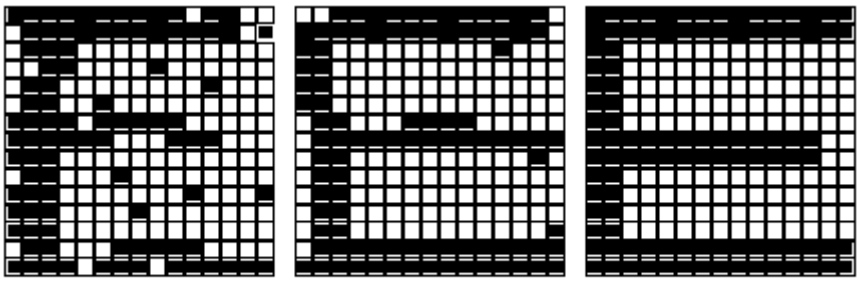
$$= -2x_k^{new}\sum_{j=1}^{n} w_{kj} x_j^{new} + 2T_k x_k^{new} + 2x_k^{old}\sum_{j=1}^{n} w_{kj} x_j^{old} - 2T_k x_k^{old}$$

$$\Delta H = 2(x_k^{old} - x_k^{new})[\sum_{j=1}^{n} w_{kj} x_j^{old} - T_k] \leq 0$$

Qualche applicazione. Abbiamo una rete di Hopfield a 256 neuroni, rappresentati ciascuno da un quadratino. Se il quadratino è nero, lo stato vale +1, altrimenti –1. Durante il processo di addestramento, la rete riceve in ingresso un certo numero di immagini (stati stabili), ognuna delle quali corrispondente ad una lettera dell'alfabeto.

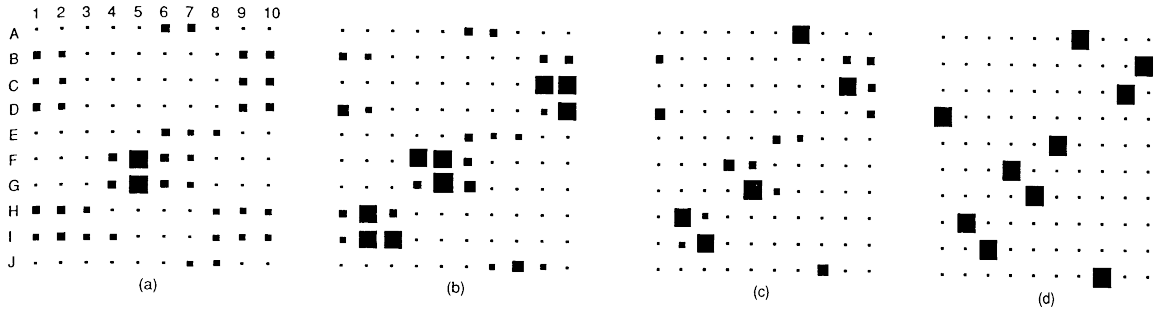
In fase operativa presentiamo alla rete una lettera 'e' 'corrotta' da un certo rumore (vedi figure che seguono). Dopo 2 e 6 iterazioni la rete raggiunge la seconda e terza configurazione mostrate nella figura; associa infine all'ingresso lo stato stabile che risulta più vicino ad esso. Il riconoscimento è andato a buon fine. L'uscita della rete non viene quindi determinata immediatamente, ma attraverso un processo iterativo, consistente nell raggiungere, a partire da uno stato iniziale, uno stato stabile finale corrispondente al minimo di una funzione di energia. Il problema è che la rete può giungere anche ad uno stato stabile 'spurio', cioè non compreso fra quelli che erano stati

memorizzati durante l'addestramento (comportamento indesiderato) (il fenomeno corrisponde al problema dei minimi locali della funzione di errore).



Stato iniziale Dopo 2 iterazioni Stato finale

Il problema del commesso viaggiatore. Abbiamo una rete 10x10, nella quale i neuroni sono però *continui* (possono assumere valori continui tra 0 (assenza di quadratino) e 1 (quadratino 'pieno')). Dalla configurazione iniziale la rete evolve verso la finale (la quarta) attraverso la minimizzazione della funzione di energia. In tale funzione dovremo inserire tutti quei vincoli che riguardano il problema del commesso viaggiatore. In sostanza, nella configurazione iniziale dovremo avere un solo quadratino pieno per ogni riga, un solo quadratino pieno per ogni colonna ed un percorso minimo che collega tutti i quadratini. Il percorso ottimo trovato dalla rete muove attraverso le città D-H-I-F-G-E-A-J-C-B.



Una rete di Hopfield può quindi essere impiegata per risolvere un problema di ottimizzazione; il problema sta nel trovare una opportuna funzione di errore da minimizzare. È interessante notare che, al contrario di quanto avviene negli algoritmi classici di programmazione lineare, per le soluzioni parziali i vincoli non sono necessariamente soddisfatti (rilassamento dei vincoli); lo sono solo per la soluzione finale[4].

[4] Demo Matlab:

DEMOSM1 – mappa auto-organizzante - al termine delle 1000 epoche, ciascuno dei 10 neuroni si è posizionato su un punto della curva assegnata - la topologia della rete riprende la topologia dell'ingresso; ciò dipende dal fatto che i neuroni erano posizionati inizialmente secondo un concetto di vicinanza - se inizialmente il neurone 5 fosse stato accanto al neurone 9, avremmo avuto una 'linea intrecciata'.

DEMOSM2 – Caso bidimensionale: una serie di punti sollecita una griglia di 5x6 neuroni, inizialmente tutti concentrati nel punto (0.5, 0.5).

DEMO/TOOLBOXES/NEURAL NETWORKS/HOPFIELD TWO NEURON DESIGN – Rete di Hopfield con due stati stabili – viene presentato in ingresso un campione, la rete risponde con uno dei due stati stabili.

DEMO/TOOLBOXES/NEURAL NETWORKS/HOPFIELD UNSTABLE EQUILIBRIA – Come nel caso precedente, ma tutti i punti sulla diagonale terminano in uno stato stabile spurio (minimi locali della funzione di energia).