sergio zani andrea cerioli

analisi dei dati e data mining per le decisioni aziendali

giuffrè editore • milano

PRIMA
CHE IL LIBRO SCIENTIFICO MUOIA

Il libro scientifico è un organismo che si basa su un equilibrio delicato.
Gli elevati costi iniziali (le ore di lavoro necessarie all'autore, ai redattori, ai compositori, agli illustratori) sono recuperati se le vendite raggiungono un certo volume.

La fotocopia in un primo tempo riduce le vendite e perciò contribuisce alla crescita del prezzo. In un secondo tempo elimina alla radice la possibilità economica di produrre nuovi libri, soprattutto scientifici.

Per la legge italiana la fotocopia di un libro (o parte di esso) coperto da diritto d'autore (Copyright) è illecita. Quindi ogni fotocopia che eviti l'acquisto di un libro è reato.

Fotocopie per uso personale del lettore possono essere effettuate nei limiti del 15% di ciascun volume dietro pagamento alla SIAE del compenso previsto dall'art. 68, comma 4, della legge 22 aprile 1941 n. 633 ovvero dall'accordo stipulato tra SIAE, AIE, SNS e CNA, CONFARTIGIANATO, CASA, CLAAI, CONFCOMMERCIO, CONFESERCENTI il 18 dicembre 2000.

Le riproduzioni ad uso differente da quello personale potranno avvenire, per un numero di pagine non superiore al 15% del presente volume, solo a seguito di specifica autorizzazione rilasciata da AIDRO, via delle Erbe n. 2, 20121 Milano, telefax 02 809506, e-mail: aidro@iol.it.

TUTTE LE COPIE DEVONO RECARE IL CONTRASSEGNO DELLA S.I.A.E.

ISBN 88-14-13695-5

Tipografia «MORI & C. S.p.A.» - 21100 Varese - Via F. Guicciardini 66

Ad Anna, alle figlie e nipoti

A Chiara, ad Alessandra ed Anna

Capitolo XII

LE RETI NEURALI

di Isabella Morlini

1. *Introduzione*

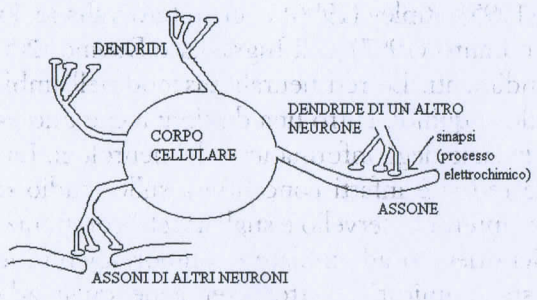
Le reti neurali artificiali (*artificial neural networks*) sono particolari modelli statistici che si ispirano al funzionamento delle reti neurali biologiche ed al modo in cui queste elaborano le informazioni, spesso anche molto complesse. La trattazione di questi modelli nel presente capitolo sarà necessariamente sintetica, per cui si rimanda a testi specifici, quali Bishop (1995), Ripley (1996) e, in ambito italiano, i volumi a cura di Bellacicco e Lauro (1997) e di Ingrassia e Davino (2002), per eventuali approfondimenti. Le reti neurali nascono nell'ambito dell'intelligenza artificiale e, quindi, entro una disciplina che può essere considerata dominio iniziale degli informatici e dei neurologi. La prima fase di sviluppo delle reti si è infatti concentrata sullo studio dei comportamenti biologici interni al cervello e sugli aspetti computazionali dei modelli elettronici orientati ad emulare i comportamenti intelligenti. Tali modelli sono stati applicati in settori eterogenei quali, ad esempio, l'ingegneria, l'informatica, l'econometria, la medicina, la finanza, le telecomunicazioni.

Ben presto, però, la spiegazione e la formalizzazione dei meccanismi di funzionamento delle reti e l'interpretazione dei risultati si sono estesi anche in ambito statistico. Di pertinenza statistica sono anche molti problemi in cui oggi vengono applicate le reti neurali: previsione dei valori futuri di serie storiche, discriminazione tra osservazioni statistiche di differenti popolazioni, formazione di gruppi omogenei di unità. L'interesse verso questi algoritmi è dettato, in particolar modo, dalle *performances* che essi offrono nei problemi di classificazione e previsione caratterizzati da informazioni incomplete, affette da errore o imprecise (si pensi, ad esempio, alle previsioni meteorologiche, alla elaborazione di immagini radar ed al riconoscimento di testi, voci ed immagini), dalla loro elevata velocità di calcolo (fondamentale, ad esempio,

nel controllo di qualità *on line* in un processo produttivo) e dalla flessibilità della forma funzionale generata.

Sebbene il parallelo tra reti neurali artificiali e comportamenti fisici e biologici interni al cervello umano sia puramente metaforico, è utile illustrare brevemente i processi che avvengono nelle strutture nervose, per poterli poi paragonare con quelli matematici elaborati da una rete neurale artificiale e rendere più chiara la spiegazione di questi ultimi. Il cervello umano è composto da cellule nervose (neuroni biologici) connesse tra loro in vario modo. Quando un neurone viene attivato da uno stimolo esterno, manda un impulso elettrochimico ai neuroni a cui è connesso (fig. 12.1). A loro volta, tali neuroni mandano uno stimolo a quelli a cui sono connessi e tale processo si ripete in modo tale che, nell'intervallo di pochi centesimi di secondo, intere regioni del cervello risultano interessate.

FIG. 12.1. *Il neurone biologico.*

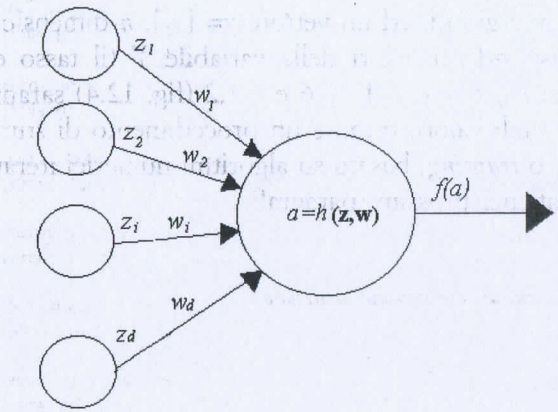


Il contenuto informativo del cervello umano non è determinato da un singolo neurone, ma dal comportamento di tutta la corteccia cerebrale: ogni singolo neurone ha compiti relativamente semplici, ed è la sua interazione con altre componenti e l'insieme dei flussi di segnali fra i vari neuroni che consentono le elaborazioni, anche altamente complesse, dell'informazione.

Il neurone artificiale è una schematizzazione di quello biologico in cui gli stati di attivazione e le relazioni sono descritti da formule matematiche e non da fenomeni elettrochimici o biologici. Il neurone artificiale viene anche detto unità o nodo. Schematicamente, il comportamento di un neurone artificiale può essere così formalizzato (fig. 12.2):

- i segnali in entrata $z_1, z_2, \ldots, z_d$ che riceve dalle d unità a cui è collegato, vengono pesati con i valori $w_1, w_2, \ldots, w_d$ che determinano il tipo di connessione tra le coppie di neuroni (positiva o negativa, forte o debole);
- lo stato di attivazione a dell'unità è dato dalla funzione $a = h(\mathbf{z},\mathbf{w})$ calcolata sul vettore $\mathbf{z} = [z_1, z_2, \ldots, z_d]'$ dei valori in entrata e su quello $\mathbf{w} = [w_1, w_2, \ldots, w_d]'$ dei pesi.
- Sullo stato di attivazione a viene poi calcolata la funzione di trasferimento $f(a)$, che determina il valore in uscita del neurone.

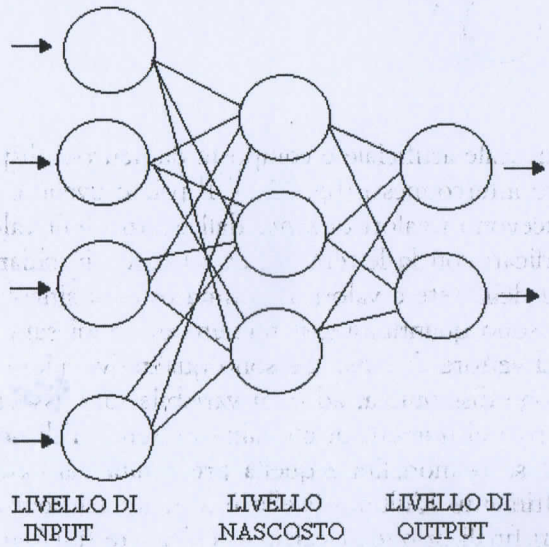
FIG. 12.2. *Il neurone artificiale.*



Una rete neurale artificiale è composta da neuroni disposti in vari livelli e tra loro interconnessi (fig. 12.3). Il primo livello è formato da neuroni che ricevono i valori di *input* dall'esterno (tali valori saranno nel seguito indicati con la lettera x, per rendere più chiara la distinzione tra *input* della rete e valori in entrata di ogni singolo neurone). Se le variabili sono quantitative, il numero dei nodi sarà uguale alla dimensione del vettore di *input*. Se sono qualitative allora, adottando una codificazione disgiuntiva, ad ogni variabile sarà associato un numero di nodi pari al numero di modalità presenti. Tali nodi assumeranno valore 1 se la modalità è quella presentata dal caso in esame, valore zero altrimenti. L'ultimo livello è formato da neuroni che forniscono i valori finali di *output* calcolati dalla rete (tali valori saranno nel seguito indicati con la lettera y, per distinguerli dai valori in uscita di ogni singolo neurone). Il numero di nodi anche in questo caso sarà pari alla dimensione del vettore di *output* se le variabili sono quantitative, alla somma delle modalità se sono qualitative. Si pensi, a titolo

di esempio, all'applicazione d'una rete neurale per la previsione del tasso di cambio dell'euro contro dollaro, sulla piazza di Milano, in base alle seguenti variabili esplicative: i tassi di cambio fra dollaro ed euro e fra euro e yen giapponese rilevati il giorno prima sulle piazze di Milano, Francoforte e New York. I neuroni di *input* saranno pari a sei e riceveranno i valori dei tassi di cambio del giorno prima. Il livello di *output* sarà costituito, invece, da un unico neurone, relativo al tasso da prevedere. Se i nodi del livello intermedio sono pari a due, l'architettura di rete sarà quella illustrata nella fig. 12.4. Data una matrice di dati $\mathbf{X}= [x_{is}]$ di dimensione $n\times 6$, in cui sono registrati i valori dei 6 tassi per n giorni, ed un vettore $\mathbf{y}= [y_i]$, n-dimensionale, contenente i corrispondenti valori della variabile Y (il tasso euro contro dollaro) i pesi w_{sj} e w_{j1}, $s=1,\ldots,6$ e $j=1,2$ (fig. 12.4) saranno determinati in base a tali valori, tramite un procedimento di stima detto addestramento, o *training*, basato su algoritmi numerici iterativi che verranno illustrati nei prossimi paragrafi.

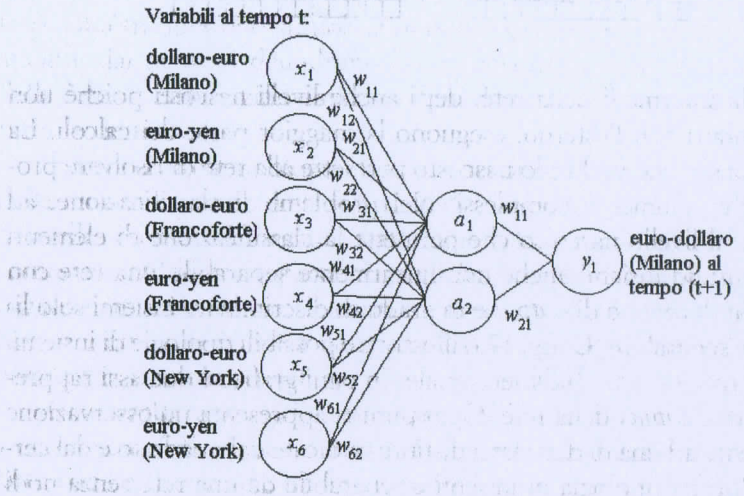
FIG. 12.3. *Esempio di rete neurale artificiale.*



Il problema posto della previsione del tasso di cambio euro-dollaro, sulla piazza di Milano, potrebbe essere affrontato anche seguendo

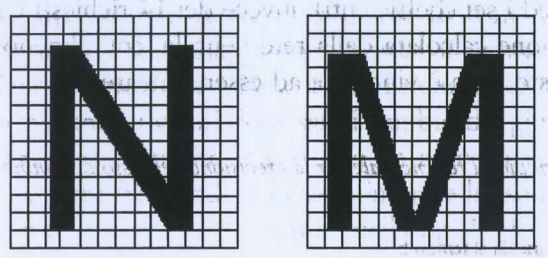
un approccio di tipo tradizionale, con un modello di regressione multipla. Tale modello richiederebbe la stima di sette parametri incogniti (l'intercetta ed i sei coefficienti), invece dei 14 richiesti dalla rete. Tuttavia la funzione calcolata dalla rete neurale, con almeno un nodo nel livello nascosto, non è vincolata ad essere lineare.

FIG. 12.4. *Esempio di rete neurale per la previsione del tasso di cambio euro contro dollaro.*



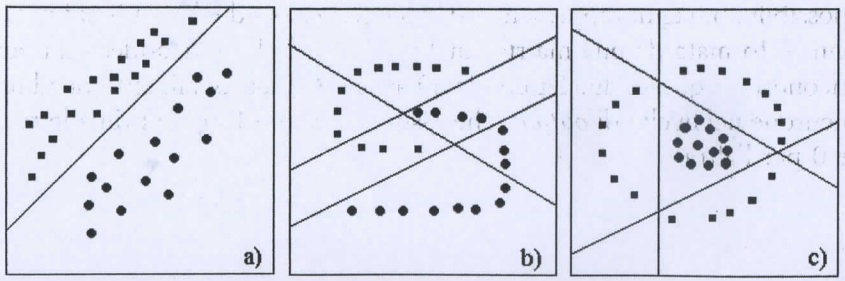
Un altro, importante, campo di applicazione riguarda il problema del riconoscimento di immagini, di forme e di testi, utilizzando reti neurali in cui ogni neurone del livello di *input* rappresenta un *pixel* (*picture element*, ovvero unità grafica elementare) che può assumere un valore compreso nell'intervallo [0, 1]: 0 se il *pixel* è completamente bianco, 1 se è completamente nero ed un valore intermedio proporzionale alla frazione del *pixel* colorata di nero, altrimenti. La fig. 12.5 illustra due possibili immagini rappresentanti le lettere «N» ed «M». Ogni immagine è formata da una matrice di 13×13 *pixels*. La rete addestrata per riconoscere queste due lettere dovrà avere 169 neuroni in entrata ed un neurone nel livello di *output* (che assumerà valore 1 per la prima lettera e 0 per l'altra).

FIG. 12.5. *Immagini composte da una matrice di 13×13 pixels il cui valore è compreso nell'intervallo [0,1] ed è proporzionale alla frazione colorata del pixel.*



I livelli intermedi della rete, detti anche livelli nascosti poiché non hanno contatti con l'esterno, eseguono la maggior parte dei calcoli. La presenza di almeno un livello nascosto permette alla rete di risolvere problemi anche altamente complessi. Nei problemi di classificazione, ad esempio, è il livello nascosto che permette la classificazione di elementi appartenenti ad insiemi anche non linearmente separabili (una rete con solo i livelli di *input* e di *output* è in grado di discriminare insiemi solo linearmente separabili). La fig. 12.6 illustra tre possibili tipologie di insiemi, per un vettore di *input* bidimensionale. In ogni grafico i due assi rappresentano i due *inputs* della rete. Ogni punto rappresenta un'osservazione appartenente ad una di due classi distinte (indicate dal quadrato e dal cerchio). La prima tipologia di insiemi è separabile da una rete senza nodi nascosti. La seconda e la terza da una rete con al minimo tre nodi nascosti. Dati che si distribuiscono nello spazio di *input* come nelle figure 12.6.b) e 12.6.c) sono difficilmente trattabili con gli algoritmi tradizionali di *cluster analysis*, ad eccezione del metodo del legame singolo.

FIG. 12.6. *Esempio di insiemi (in spazi bidimensionali) separabili a) da una rete senza nodi nascosti e b) e c) da una rete con al minimo tre nodi nel livello nascosto*



Il compito della rete è quello di calcolare il vettore di *output* ogni volta che viene presentato un vettore di *input*, in altre parole calcolare la funzione $y=f(\mathbf{x})$ da R^p ad R^c, dove p è il numero dei neuroni del primo livello e c è il numero di neuroni dell'ultimo livello. Il calcolo della funzione avviene nel modo seguente: quando pervengono dall'esterno i dati di *input*, le unità del primo livello si attivano e mandano tali segnali ai neuroni a cui sono collegate. Questi ultimi ricevono un valore che dipende dal tipo di connessione (ovvero dal peso w e dalla funzione di trasferimento) e quindi sono attivati in maniera diversa. A loro volta trasmettono stimoli ai nodi cui sono collegati e le attivazioni assunte dai neuroni dell'ultimo livello costituiscono i valori di *output* della rete. L'elaborazione avviene, quindi, attivando in modo parallelo tutte le unità.

Esistono molte tipologie di rete, a seconda del tipo di connessioni, della funzione $h(\mathbf{z};\mathbf{w})$, della funzione di trasferimento $f(a)$ e del numero di livelli nascosti. Nel seguito tratteremo solo le architetture di rete più comunemente usate, in particolare quelle a tre o a due livelli, *totalmente connesse* (in cui ogni unità è connessa con tutte quelle del livello successivo, ma non con quelle dello stesso livello) e *non ricorrenti* (con connessioni in un solo senso).

Una caratteristica delle reti neurali è che esse sono robuste rispetto a dati incompleti, incerti o affetti da errore. Tale peculiarità deriva dal fatto che l'elaborazione finale è la somma delle molte elaborazioni parziali che avvengono nei singoli neuroni e che quindi un errore presente in un'unità non influisce in modo sensibile sul risultato finale. Ne consegue che le reti neurali sono particolarmente vantaggiose in quei problemi di classificazione e previsione che richiedono soprattutto robustezza piuttosto che precisione.

Una seconda caratteristica è che il processo di stima dei parametri (dei pesi) avviene attraverso una fase di addestramento. Durante questa fase viene presentato alla rete un insieme di osservazioni (dette anche «esempi»). La rete apprende le relazioni presenti nei dati solo in base agli esempi e senza che le vengano fornite indicazioni riguardo alla forma funzionale del processo generatore o alla distribuzione di probabilità dei dati. Per tale ragione le reti risultano particolarmente appropriate anche in quei campi applicativi in cui non si riesca o non sia conveniente trovare una soluzione algebrica o algoritmica: si pensi, ad esempio, alle previsioni meteorologiche, al riconoscimento di immagini e alla diagnostica medica (si veda, ad esempio, Morlini e Orlandini (2001), per un'applicazione sulle immagini radar). Di seguito verrà

chiarito, in modo formale, come avvenga tale processo di apprendimento. In maniera generica si può sottolineare come l'idea di base dell'apprendimento sia il rafforzamento delle connessioni tra due unità, entrambe attive, ogni volta che un esempio viene presentato alla rete.

L'apprendimento (*training* o *learning*) può essere di tipo *supervisionato* o *non supervisionato*. Nel primo caso alla rete vengono forniti sia i dati di *input* sia quelli di *output* (che chiameremo *target*, per distinguerli dall'*output* effettivo calcolato dalla rete). Nel secondo caso vengono specificati solo i valori di *input* e la rete non apprende più la funzione $\mathbf{y}=f(\mathbf{x})$ ma organizza e classifica le osservazioni trovando relazioni e regolarità tra esse (come avviene, ad esempio, nella *cluster analysis*, nello *scaling multidimensionale* e nell'analisi in componenti principali). La fase di apprendimento è ben distinta da quella di esecuzione. Durante la fase di apprendimento vengono trovati i valori ottimali dei pesi $\mathbf{w}$. Durante la fase di esecuzione viene determinato il vettore di *output* $\mathbf{y}$, dati $\mathbf{x}$ e $\mathbf{w}$, se l'apprendimento è supervisionato, oppure il neurone o il gruppo di appartenenza se l'apprendimento è non supervisionato.

All'interno del *package* statistico *SPSS*, le reti neurali artificiali che tratteremo nel seguito di questo capitolo sono disponibili nel pacchetto *Neural Connection*. In Matlab sono invece disponibili nel Toolbox "nntool".

2. *Il* multi-layer perceptron (MLP)

Il *multi-layer perceptron* (percettrone multistrato) è la rete neurale maggiormente conosciuta ed utilizzata nei problemi di previsione e di classificazione in cui sono noti, per un insieme di osservazioni (denominato *training* o *learning set*), sia i valori delle variabili esplicative sia quelli delle variabili dipendenti (*target*). La tecnica di addestramento utilizzata per trovare i valori ottimali dei pesi che compaiono nella rete si chiama *back propagation*. Tale tecnica procede in maniera iterativa, partendo da valori iniziali dei pesi. Ogni iterazione viene chiamata «epoca». Ad ogni epoca le osservazioni del *training set* vengono sottoposte ad una procedura che prevede due fasi dette, rispettivamente, *forward pass* e *backward pass*. Nel *forward pass* (fase di esecuzione) la rete trova i valori di $\mathbf{y}=f(\mathbf{x})$ dati $\mathbf{x}$ e $\mathbf{w}$. Nel *backward pass* (fase di apprendimento) trova $\mathbf{w}$ dati $\mathbf{y}$ ed $\mathbf{x}$.

2.1. *Il* forward pass

Consideriamo una rete neurale con p nodi nel livello di *input*, m nodi nel livello nascosto e c nodi nel livello di *output*. Consideriamo, inoltre, un *training set* costituito da n osservazioni e indichiamo con i il generico «esempio» ad esso appartenente $i=1,2,\ldots,n$. Quando i valori di *input* relativi alla i-esima osservazione vengono inseriti nella rete, i nodi del primo livello trasmettono tali valori a ciascuna delle m unità del secondo livello. Queste unità calcolano la combinazione lineare degli *inputs* ricevuti, usando come coefficienti i valori correnti dei pesi. Con riferimento al j-esimo neurone del livello nascosto $j=1,2,\ldots,m$ si avrà quindi:

$$a_j = h(\mathbf{x}, \mathbf{w}) = w_{0j} + \sum_{s=1}^{p} w_{sj} x_{is} \qquad (12.1)$$

dove a_j indica lo stato di attivazione del nodo j, x_{is} è il valore relativo alla s-esima variabile di *input* nella i-esima osservazione, w_{sj} è il peso che lega l's-esimo neurone di *input* con il j-esimo neurone del livello nascosto e w_{0j} è una costante, detta *bias*, riferita al livello di *input*. Il valore di tale costante è stimato dalla rete in modo analogo agli altri parametri: questo suggerisce di considerare il *bias* come il peso di una connessione fittizia da una unità che vale sempre uno. Supponendo di avere p unità nel primo livello, il *bias* sarà realizzato attraverso una $(p+1)$-esima unità il cui valore è fisso ad uno e la formula potrà essere riscritta nel modo seguente:

$$a_j = h(\mathbf{x}, \mathbf{w}) = \sum_{s=1}^{p+1} w_{sj} x_{is} \qquad (12.2)$$

Il valore in uscita del j-esimo neurone nascosto è ottenuto applicando ad a_j una funzione di trasferimento, scelta a seconda delle ipotesi adottate. La prima forma storicamente usata e quella che meglio esprime l'idea di stati bipolari, è la funzione a gradino

$$f(a_j) = \begin{cases} 1 & \text{se } a_j > th \\ 0 & \text{se } a_j \leq th \end{cases} \qquad (12.3)$$

dove th (*threshold*) è un valore «soglia» determinato dal ricercatore. Essendo discontinua, tale funzione è spesso causa di complicazioni formali e questo giustifica il ricorso a funzioni sigmoidi (monotone crescenti, limitate superiormente ed inferiormente, continue e regolari e

quindi ovunque differenziabili). La più comunemente usata è la funzione logistica

$$f(a_j) = \frac{1}{1 + e^{-a_j}} \qquad (12.4)$$

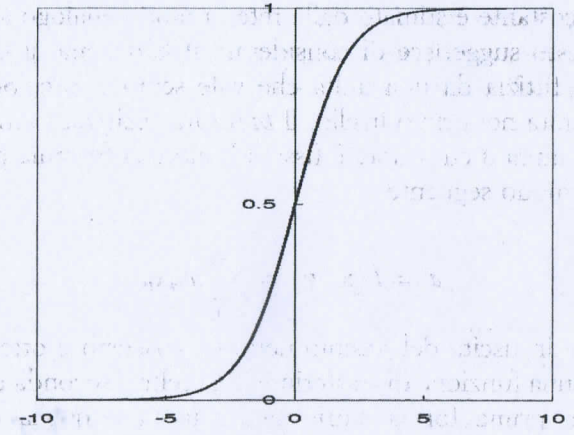
rappresentata graficamente nella fig. 12.7. Tale funzione ha come dominio l'asse reale e come codominio l'insieme (0,1). Assume valore 0.5 per $a_j = 0$.

Funzioni sigmoidi che variano nell'intervallo (-1,1) sono invece le seguenti:

$$f(a_j) = \tanh(a_j) = \frac{e^{a_j} - e^{-a_j}}{e^{a_j} + e^{-a_j}} \qquad (12.5)$$

$$f(a_j) = 2\frac{1}{1 + e^{-a_j}} - 1 = \frac{1 - e^{-a_j}}{1 + e^{-a_j}} \qquad (12.6)$$

FIG. 12.7. *Funzione logistica.*



La funzione di trasferimento logistica può essere definita in termini di **x**. A fini didattici, consideriamo un solo valore di *input* x, e quindi $a_j = w_{0j} + w_{1j}x$ (stato di attivazione del j-esimo neurone nascosto). Quando $w_{0j} = 0$ e $w_{1j} = 1$, allora $f(a_j) = f(x)$ ed il grafico coincide con quello della fig. 12.7. Quando $w_{0j} = 0$ e $w_{1j} = -1$, l'andamento della curva viene invertito (fig. 12.8). Variando w_{1j} tra 1 e 2, la pendenza della curva nel punto x=0 varia da 0.25 a 0.5 (fig. 12.9). Mantenendo invece costante w_{1j} e cambiando il valore del peso w_{0j}, cambia la posizione della curva sull'asse delle x (fig. 12.10). Dunque, i pesi w_{0j} e w_{1j} controllano, rispettivamente, la pendenza ed il valore della funzione logistica in corrispondenza di $x = 0$.

FIG. 12.8. *Funzione logistica di (-x)*

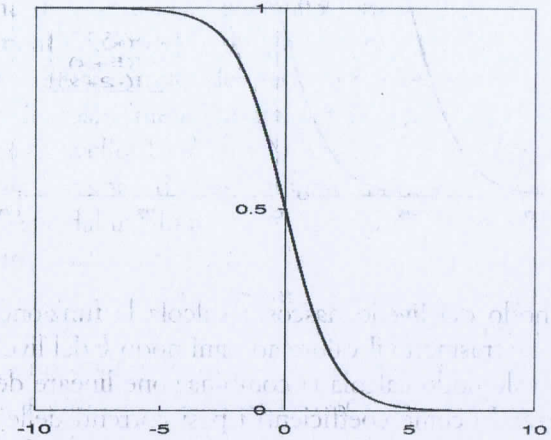


FIG. 12.9. *Funzioni logistiche di x, 2x e 0.2x.*

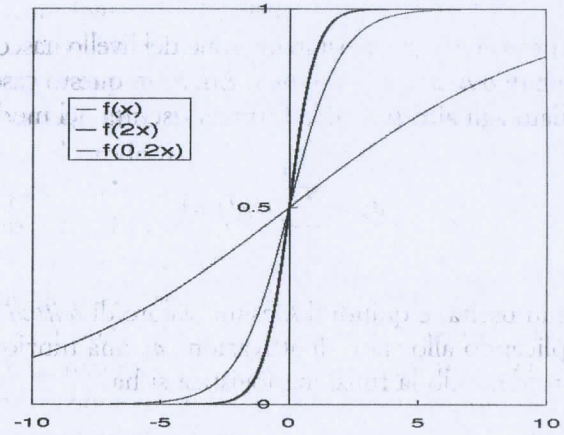
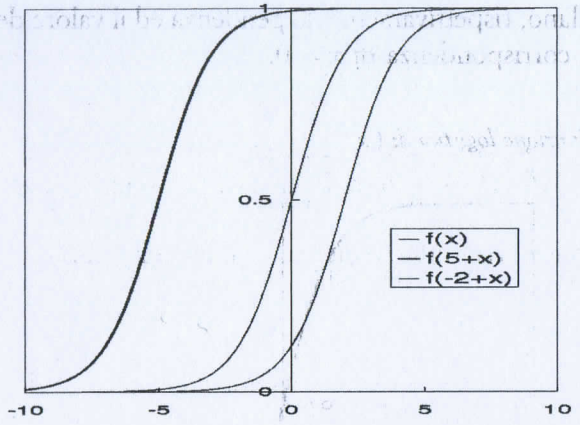


FIG. 12.10. *Funzioni logistiche di x,(5+x),(−2+x).*



Ciascun nodo del livello nascosto calcola la funzione di trasferimento su a_j e ne trasmette il valore ad ogni nodo k del livello di *output* $k = 1, \ldots, c$. Tale nodo calcola la combinazione lineare degli m valori ricevuti utilizzando come coefficienti i pesi correnti delle connessioni con il livello nascosto. Lo stato di attivazione relativo al k-esimo neurone di *output* è quindi:

$$a_k = w_{0k} + \sum_{j=1}^{m} w_{jk} f(a_j) \tag{12.7}$$

dove w_{jk} è il peso che lega il j-esimo neurone del livello nascosto con il k-esimo neurone di *output* e w_{0k} è il *bias*. Anche in questo caso il *bias* può essere assimilato agli altri pesi, e la formula riscritta nel modo seguente:

$$a_k = \sum_{j=1}^{m+1} w_{jk} f(a_j) \tag{12.8}$$

Il valore in uscita, e quindi il k-esimo valore di *output* della rete, è ottenuto applicando allo stato di attivazione a_k una funzione di trasferimento. Considerando la funzione logistica si ha

$$y_{ik} = \frac{1}{1 + e^{-a_k}} \tag{12.9}$$

dove y_{ik} è il k-esimo *output* della rete in corrispondenza dell'i-esimo vettore di *input* x_i. La stessa formula può essere riscritta in funzione dello stesso x, esplicitando i passi precedentemente descritti:

$$y_{ik} = f\left(\sum_{j=1}^{m+1} w_{jk} f\left(\sum_{s=1}^{p+1} w_{sj} x_{ij}\right)\right) \tag{12.10}$$

Alla fine di questa fase, chiamata *forward pass*, si ottengono, per ciascun elemento del *training set*, gli *outputs* effettivi della rete, prodotti sulla base dei valori correnti dei pesi. Non è necessario, in generale, che la funzione di trasferimento usata per il secondo livello sia uguale a quella del terzo livello. Inoltre, non è essenziale per i neuroni di *output* applicare una funzione di trasferimento alla combinazione lineare dei valori che riceve dal livello nascosto: questa stessa combinazione lineare può costituire l'*output* effettivo della rete, come previsto nel pacchetto *Neural Connection*, che non consente la scelta di una funzione sigmoide nell'ultimo livello. Tale limitazione può comportare un aumento del numero delle iterazioni necessarie per raggiungere la convergenza, ma non pregiudica la *performance* finale della rete e, soprattutto, consente di ottenere valori in tutto l'asse reale e non solo negli intervalli (0,1) o (-1,1).

Per chiarire cosa effettivamente avviene durante la fase di esecuzione della rete, consideriamo una semplicissima applicazione, in cui viene calcolata la funzione logica «AND» (date due asserzioni, che chiameremo asserzione «A» ed asserzione «B», l'unione di A e B tramite l'operatore logico AND risulta vera se e solo se A e B sono entrambe vere). Consideriamo due nodi di *input* rappresentanti, ad esempio, i fattori del prodotto di due termini (interi e positivi). Sia 1 il valore che essi assumono se il numero è dispari e 0 se è pari. Il risultato è uguale ad 1 se il prodotto è un numero dispari, a 0 in caso contrario. Le quattro possibili alternative contemplate dalla funzione sono elencate nella tab. 12.1. Tale funzione può essere realizzata da una semplice rete, senza livelli intermedi, con due nodi di *input*, un nodo di *output* ed una funzione di trasferimento a gradino tra il primo ed il secondo livello, con valore soglia pari a 1. Se i pesi delle connessioni sono tutti uguali a 0.5 (l'architettura di rete è mostrata nella fig. 12.11), allora lo stato di attivazione del nodo di *output* è maggiore di uno (e quindi la funzione di trasferimento uguale a 1) se e solo se entrambi gli *inputs* sono uguali ad uno (e quindi la loro combinazione lineare con pesi 0.5, considerando anche la costante uguale a 0.5, è 1.5):

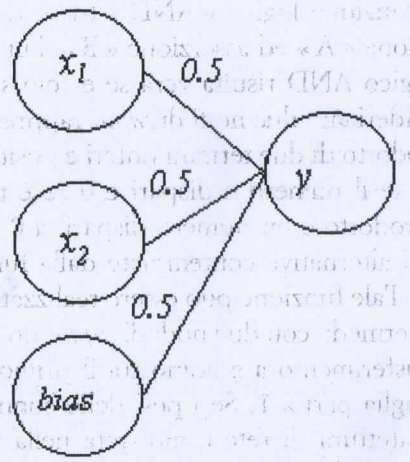
$$y(x_1) = y(1,1) = f(0.5 + 0.5\times1 + 0.5\times1) = f(1.5) = 1$$
$$y(x_2) = y(1,0) = f(0.5 + 0.5\times1 + 0.5\times0) = f(1) = 0$$
$$y(x_3) = y(0,1) = f(0.5 + 0.5\times0 + 0.5\times1) = f(1) = 0$$
$$y(x_4) = y(0,0) = f(0.5 + 0.5\times1 + 0.5\times1) = f(0.5) = 0$$

Tornando al caso generale, esistono una serie di studi sulle possibilità delle reti di rappresentare, o anche solo approssimare, funzioni definite sui medesimi domini.

TAB. 12.1. *Funzione logica AND.*

primo input	secondo input	target
1	1	1
1	0	0
0	1	0
0	0	0

FIG. 12.11. *Architettura di rete MLP per la risoluzione del problema logico «AND».*



Diversi teoremi (si vedano, ad esempio, White (1989), Hornik *et al.* (1990)) affermano che una qualsiasi funzione $y=f(\mathbf{x})$ può essere approssimata con un qualsiasi grado di accuratezza da una rete neurale a soli tre livelli, non ricorrente, avente un opportuno numero di unità nel livello nascosto, con funzione sigmoide fra il primo ed il secondo livello e funzione identità fra il secondo ed il terzo. Questi teoremi rivestono un'importanza teorica fondamentale, in quanto negano l'esistenza di problemi che non possano essere risolti da una rete con un solo livello nascosto (argomentando, al contrario, l'inutilità di più livelli nascosti) e attribuiscono le eventuali cause di insuccesso nelle applicazioni a un numero di unità nascoste non adeguato o all'inesistenza d'una relazione stabile tra *input* ed *output*. Tuttavia, dal punto di vista pratico, tali studi non indicano quante debbano essere le unità del livello nascosto e quali i pesi. Il numero dei nodi determina la flessibilità della funzione calcolata dalla rete rimane una decisione soggettiva dell'operatore, di solito determinata dal grado di complessità dalla funzione da approssimare.

2.2. *Il* backward pass

Idealmente, i pesi della rete possono essere visti come i coefficienti d'una *mapping function*, funzione le cui variabili esplicative sono gli *inputs* della rete e le cui variabili dipendenti sono gli *outputs*. La *mapping function* prodotta dalla rete neurale è molto flessibile e, come sottolineato nel paragrafo precedente, può essere configurata in modo da approssimare una qualsiasi funzione *target*, non nota, con qualsiasi grado di accuratezza. Il problema è la determinazione dei valori ottimali dei pesi, determinazione che avviene durante la fase di *backward pass*. Una volta ottenuti, nella fase *forward*, i valori di *output* relativi ad ogni vettore di *input* del *training set*, segue il confronto con i valori di *target* (valori corretti) che, ricordiamo, per gli elementi del *training set* sono supposti noti. Sulla base dell'informazione dell'errore commesso, che scaturisce da tale confronto, ogni nodo di *output* individua la direzione in cui i suoi pesi dovrebbero variare al fine di ridurre l'errore e determina l'entità del cambiamento. Quindi, ogni neurone trasmette al livello precedente l'informazione relativa all'errore commesso. Ciascun nodo del livello nascosto utilizza l'informazione trasmessa dal livello di *output* e, a sua volta, determina la direzione in cui i suoi pesi dovrebbero variare per ridurre l'errore commesso e l'entità del cambiamento. Il confronto tra *outputs* effettivi della rete e valori di *target* mira a valutare l'adattamento della *mapping function* agli elementi che costituiscono il *training set*. Epoca dopo epoca, il processo di modifica dei pesi

si basa proprio sull'ottimizzazione di una misura di tale adattamento. La misura più comunemente usata è l'errore quadratico:

$$E = \frac{1}{2}\sum_{i=1}^{n}\sum_{k=1}^{c}(y_{ik} - t_{ik})^2 \qquad (12.11)$$

dove t_{ik} è il k-esimo valore di *target* relativo all'i-esima osservazione ed y_{ik} è il k-esimo valore di *output* calcolato dalla rete. Il ricorso a tale errore può essere giustificato seguendo un approccio probabilistico. Se si considerano le distribuzioni condizionate $p(\mathbf{t}|\mathbf{x}_i)$ dove $\mathbf{t}$ è la variabile aleatoria *target* e $\mathbf{x}_i$ è la variabile aleatoria i-esimo *input* della rete, e si specificano le ipotesi di normalità, omoschedasticità ed incorrelazione di tali distribuzioni, sia rispetto ai valori di $\mathbf{x}_i$, sia rispetto alle componenti t_k del vettore $\mathbf{t}$, allora la minimizzazione dell'errore quadratico conduce alla stima di massima verosimiglianza per i pesi della rete. La somma degli errori al quadrato non rappresenta l'unica possibilità. Esistono anche altre misure di adattamento che, in relazione ai problemi affrontati, mostrano proprietà anche maggiormente auspicabili (ad esempio, l'entropia per problemi di classificazione). Tali misure non verranno trattate in questa sede e, per una loro dettagliata e completa descrizione, si rimanda a Bishop (1995), capitolo VI.

Se la rete considerata non avesse il livello nascosto ma solo i livelli di *input* e di *output* e se al posto delle funzioni di trasferimento sigmoidi si avessero delle funzioni lineari, allora la somma degli errori al quadrato sarebbe una funzione quadratica dei pesi e il problema della minimizzazione potrebbe essere ricondotto alla risoluzione d'un sistema di equazioni lineari (come avviene nella stima dei parametri di una regressione lineare multipla). Per reti maggiormente complesse, invece, al fine di risolvere il problema della minimizzazione, è necessario ricorrere ad algoritmi che prevedono l'aggiustamento iterativo dei pesi. L'algoritmo classico è noto con il nome di *back-propagation*: esso è caratterizzato dal fatto che, in ogni epoca, i cambiamenti nei pesi si muovono nella cosiddetta direzione di *steepest descent*, ovvero nella direzione in cui l'errore si riduce più rapidamente. Per una rassegna sui metodi di ottimizzazione di funzioni non lineari si rimanda a Thisted (1988) e Gentle (1998). L'identificazione di questa direzione richiede il calcolo delle derivate parziali dell'errore rispetto ai pesi, calcolo che avviene grazie alla propagazione dell'errore dal livello di *output* a quello intermedio e da quello intermedio al livello di *input*. Dato il *training set*, l'errore E dipende dal vettore $\mathbf{w}$ dell'insieme dei pesi della rete, e quindi

$E=E(\mathbf{w})$. La funzione di errore $E(\mathbf{w})$ può essere interpretata geometricamente come una superficie nello spazio dei pesi in cui ogni punto rappresenta ogni possibile vettore $\mathbf{w}$. Sebbene non sia computazionalmente fattibile determinare la forma della superficie dell'errore, è possibile conoscere la sua altezza e la sua pendenza in corrispondenza di ogni punto. Tale pendenza è data dalla direzione e dal tasso al quale l'errore della rete varia al variare di ogni peso. In termini matematici, la pendenza in ogni punto è determinata dalle derivate parziali dell'errore rispetto a ciascun peso.

2.2.1. *Le derivate dell'errore*

La derivabilità dell'errore quadratico rispetto ai pesi discende, congiuntamente, dalla differenziabilità delle funzioni sigmoidi rispetto ai pesi e dalla differenziabilità della somma degli errori al quadrato rispetto ai valori di *output* della rete. Consideriamo, inizialmente, le derivate parziali rispetto ad un generico peso: queste possono essere viste come somma delle derivate parziali calcolate su ciascun elemento del *training set*. Infatti l'errore può essere scomposto nella maniera seguente:

$$E = \frac{1}{2}\sum_{i=1}^{n}\sum_{k=1}^{c}(y_{ik} - t_{ik})^2 = \frac{1}{2}\sum_{k=1}^{c}(y_{1k} - t_{1k})^2 + \frac{1}{2}\sum_{k=1}^{c}(y_{2k} - t_{2k})^2 + \ldots + \frac{1}{2}\sum_{k=1}^{c}(y_{nk} - t_{nk})^2 \qquad (12.12)$$

da cui

$$E = E_1 + E_2 + \ldots + E_i + \ldots + E_n \qquad (12.13)$$

dove

$$E_i = \frac{1}{2}\sum_{k=1}^{c}(y_{ik} - t_{ik})^2 \qquad i = 1, \ldots, n \qquad (12.14)$$

Quindi, l'errore complessivo ad una certa epoca può essere visto come la somma di tanti contributi quante sono le osservazioni del *training set*. Analogamente, la derivata dell'errore rispetto ad un generico peso w può essere vista come somma di n derivate parziali:

$$\frac{\partial E}{\partial w} = \sum_{i=1}^{n}\frac{\partial E_i}{\partial w} \qquad (12.15)$$

Tale derivata è calcolata moltiplicando tra loro una serie di fattori corrispondenti ai passi attraverso i quali il peso considerato contribuisce all'errore. Ad esempio, un peso relativo al legame tra un neurone di *input* ed un neurone del livello nascosto incide, in prima istanza ed in maniera diretta, sullo stato si attivazione del nodo nascosto; poi sul valore della funzione di trasferimento calcolata su tale attivazione; quindi sul valore in entrata di tutti i nodi del livello di *output* e, infine, sui valori in uscita. Una modifica su un peso di un nodo nascosto causa, quindi, cambiamenti lungo tutto il processo descritto. Percorrendo tale «catena» in maniera inversa, si giunge all'espressione della derivata dell'errore rispetto al peso considerato (da qui il nome di *chain rule* riferito alla legge che governa il calcolo di tale derivata).

L'espressione analitica della derivata rispetto al generico peso w_{jk} della connessione tra il j-esimo neurone del livello nascosto ed il k-esimo neurone di *output*, è la seguente:

$$\frac{\partial E_i}{\partial w_{ik}} = (y_k - t_k)y_k(1 - y_k)a_j \qquad (12.16)$$

Quella rispetto al peso w_{sj} della connessione tra l's-esimo nodo di *input* ed il j-esimo nodo del livello nascosto è:

$$\frac{\partial E_i}{\partial w_{sj}} = [(y_k - t_k)y_k(1 - y_k)a_j]f(a_j)(1 - f(a_j))x_s \qquad (12.17)$$

2.2.2. *Le regole di apprendimento*

Una volta determinate le derivate parziali dell'errore rispetto ad ogni peso della rete, sommando le derivate parziali riferite ad ogni esempio nel *training set*, i neuroni della rete modificano i loro pesi sulla base d'una determinata regola di apprendimento. La regola di apprendimento usata nella versione classica della *back-propagation* è lo *steepest descent*, che determina i cambiamenti nella direzione in cui l'errore si riduce più rapidamente. Lo spazio dei pesi della rete ha tante dimensioni quanti sono i pesi, ed ogni punto in tale spazio corrisponde ad un vettore di $((p+1)\times m + (m+1)\times c)$ elementi. Durante l'addestramento, la rete passa attraverso alcuni di questi punti, in corrispondenza dei quali l'errore raggiunge un certo valore, che determina l'altezza della superficie. L'altezza della superficie dell'errore non è nota per tutti i punti dello spazio, ma solo per quelli che corrispondono ai vettori dei pesi attraverso i quali la rete passa effettivamente. In corrispondenza di tali punti l'algoritmo individua la direzione in cui la superficie decresce più rapidamente e determina l'entità del cambiamento dei pesi in tale direzione. In termini matematici, ad ogni passo τ di un processo iterativo, la direzione è quella data dal gradiente calcolato sul vettore $\mathbf{w}(\tau)$, considerato con segno negativo, e l'entità del cambiamento è quella determinata da un parametro η, fissato dal ricercatore:

$$\Delta\mathbf{w}(\tau) = -\eta\ \nabla E|\mathrm{w}(\tau) \qquad (12.18)$$

dove Δ è l'operatore differenza e ∇ è l'operatore gradiente ($\nabla E|_{w(\tau)}$ indica il vettore le cui componenti sono le derivate parziali dell'errore E rispetto ad ogni singolo elemento $w(\tau)$ del vettore $\mathbf{w}(\tau)$). Ad ogni passo τ, ogni singolo peso w viene modificato nel modo seguente:

$$\Delta w(\tau) = -\eta\frac{\partial E}{\partial w(\tau)} \qquad (12.19)$$

dove E indica l'errore calcolato su tutto il *training set* ed il parametro η, chiamato *learning rate*, controlla l'entità del cambiamento del peso. In genere il valore di η viene definito nell'intervallo (0,1). Se η è troppo elevato, l'algoritmo può determinare cambiamenti nei pesi che portano ad un aumento dell'errore, piuttosto che ad un decremento, e ad oscillazioni divergenti tali per cui la condizione di convergenza ΔE=0 non sarà mai soddisfatta. Viceversa, se il valore è troppo piccolo, la ricerca del vettore dei pesi ottimale può procedere in modo estremamente lento, implicando tempi computazionali molto lunghi. Il valore appropriato di η, che può variare a seconda del problema o, all'interno di uno stesso problema, a seconda del *training set* selezionato, è generalmente scelto in maniera euristica, addestrando più volte la rete con differenti valori e analizzandone il comportamento. Esistono varianti dello *steepest descent* che mirano ad automatizzare la scelta del parametro η, considerandola parte integrante dell'algoritmo di apprendimento. Una di queste tecniche è, ad esempio, la *bold-driver technique*. Ad ogni passo del processo iterativo essa determina il nuovo valore η(new) del *learning rate* sulla base di quello precedente η(old). In termini formali:

$$\eta(\text{new}) = \begin{cases} \rho\eta(\text{old}) & \text{se } \Delta E < 0 \\ \sigma\eta(\text{old}) & \text{se } \Delta E > 0 \end{cases} \qquad (12.20)$$

con ρ leggermente superiore all'unità (ad esempio, pari ad 1.1) e σ significativamente inferiore all'unità (ad esempio, pari a 0.5). Altre varianti sono state proposte in letteratura. Esse prevedono un differente valore di η per ciascun peso della rete, che viene aggiornato durante la fase di addestramento. La regola dello *steepest descent* viene così modificata:

$$\Delta w(\tau) = w(\tau+1) - w(\tau) = -\eta(\tau)\frac{\partial E}{\partial w(\tau)} \qquad (12.21)$$

Per un approfondimento sui metodi proposti per aggiornare il *learning rate* dei diversi pesi si veda, ad esempio, Smith (1996).

Esistono parecchie varianti all'algoritmo originale dello *steepest descent*. Una di queste è il cosiddetto *example-based learning*, che effettua le modifiche sui pesi dopo aver calcolato le derivate parziali dell'errore su un singolo elemento del *training set* (al contrario della versione originale, che effettua i cambiamenti alla fine di ogni epoca, ovvero dopo aver calcolato le derivate su tutti gli elementi del campione). Tale variante comporta, all'inizio, una riduzione dell'errore più veloce, poiché la rete non deve aspettare un'intera epoca prima di poter adattare i pesi, ma non implica necessariamente un tempo di addestramento inferiore. Le operazioni di calcolo complessive sono, infatti, più numerose, dovendo i pesi essere modificati dopo ogni singolo esempio. In seconda istanza, tale algoritmo risente dell'ordine in cui vengono presentate alla rete le osservazioni: quelle presentate per ultime hanno, infatti, un maggior impatto rispetto a quelle presentate per prime. Per tale motivo gli esempi devono essere proposti alla rete in un ordine casuale, che non rispetti le loro proprietà statistiche (idealmente, anche in un ordine diverso per ogni epoca). In ultimo, l'adattamento dei pesi dopo il calcolo di ogni singolo gradiente comporta un andamento estremamente oscillatorio (spesso, ciò che viene «appreso» da una osservazione, viene completamente perso con la successiva). Al contrario, l'apprendimento per epoche comporta un cambiamento dettato da un valore medio e, quindi, meno variabile. Una forma di compromesso è il cosiddetto *batch-learning*, che effettua le modifiche sui pesi sulla base di un numero k di esempi, determinato a priori dal ricercatore.

Una seconda versione dello *steepest descent* è lo *steepest descent plus momentum* che implica, nella formula della modifica dei pesi, un ulteriore parametro, detto momento e indicato con μ, compreso nell'intervallo (0,1):

$$\Delta w(\tau) = -(1-\mu)\eta\frac{\partial E}{\partial w(\tau)} + \mu\Delta w(\tau-1) \qquad (12.22)$$

Tale parametro conferisce alle fasi finali dell'addestramento una maggior inerzia al movimento nello spazio dei pesi, poiché le fasi precedenti intervengono ad influenzare i successivi cambiamenti, anche se in misura sempre minore, e l'algoritmo evita le brevi oscillazioni temporanee. Un cambiamento in una certa direzione deve, infatti, persistere per un certo periodo di tempo (proporzionale al valore di μ) prima di poter influenzare la direzione in cui i pesi vengono effettivamente cambiati. L'utilità del momento non è vista in maniera unanime. Ad esempio, Bishop (1995) afferma come tale parametro porti sempre ad una convergenza dell'algoritmo più veloce e senza oscillazioni divergenti. Smith (1996), al contrario, ritiene che esso possa risultare utile solo per determinate forme della superficie di errore. Il valore di μ, al pari di η, deve essere determinato in modo empirico, sulla base del *training set*.

Un algoritmo di apprendimento proposto da Fahlman (1989) per diminuire i tempi computazionali dello *steepest descent*, è il metodo *quickprop*. La formula relativa alla modifica dei pesi è la seguente

$$\Delta w(\tau) = \frac{(\partial E/\partial w)(\tau)}{(\partial E/\partial w)(\tau-1) - (\partial E\backslash\partial w)(\tau)}\Delta w(\tau-1) \qquad (12.23)$$

Tale algoritmo implica un elevato livello di soggettività nella scelta del punto iniziale del processo e del comportamento da attuare quando $\Delta w(\tau\text{-}1)=0$, ovvero quando la derivata corrente $(\partial E/\partial w)(\tau)$ risulta maggiore (e con lo stesso segno) della derivata al passo precedente $(\partial E=\partial w)(\tau\text{-}1)$. Il concetto essenziale alla base di questo metodo è l'utilizzo della conoscenza riguardo alla curvatura della superficie di errore.

La regola di apprendimento più comunemente usata, dopo lo *steepest descent*, è nota con il nome di *conjugate gradient* (anche chiamata *secant method*). Tale algoritmo misura il gradiente della superficie di errore epoca dopo epoca e modifica i pesi in una direzione di compromesso tra quella di *steepest descent* e quella precedente. Nel caso in cui la funzione di errore abbia curvature sostanzialmente diverse lungo differenti direzioni, in corrispondenza della maggior parte dei punti dello spazio dei pesi il gradiente locale negativo $-\nabla E$ non punta direttamente verso il minimo della superficie di errore. Pertanto, l'uso dello *steepest descent* può comportare movimenti oscillatori dei pesi con pic-

coli progressi verso il minimo e, quindi, sia tempi computazionali notevolmente lunghi prima di raggiungere il minimo globale, sia il rischio di rimanere «imprigionati» in un minimo locale. Il metodo del *conjugate gradient* cerca di ovviare a tali inconvenienti. Con riferimento ad un singolo peso, oltre alla derivata prima, che determina la pendenza dell'errore in un certo punto e quindi l'entità del cambiamento nel peso, esso misura anche la derivata seconda. Tale derivata misura proprio la curvatura della superficie nel punto e quindi l'entità del cambiamento della pendenza. In generale, il cambiamento ottimale in un peso, quello richiesto affinché la curvatura abbia una pendenza nulla, può essere stimato dividendo la derivata prima per la seconda:

$$\Delta w(\tau) = \frac{(\partial E/\partial w)}{(\partial^2 E/\partial w^2)} \qquad (12.24)$$

Poiché il calcolo preciso della derivata seconda è praticamente impossibile, l'algoritmo prevede valori approssimati di tale derivata.

La trattazione delle regole di apprendimento qui riportata non è certamente esaustiva. Si è focalizzata l'attenzione su quegli algoritmi più comunemente usati e presenti nei principali pacchetti statistici (il pacchetto *Neural Connection*, ad esempio, offre la possibilità di scegliere fra lo *steepest descent plus momentum* e il *conjugate gradient,* mentre il Toolbox sulle reti neurali di Matlab propone, oltre a questi, più di un'altra decina di regole di apprendimento). Per gli altri algoritmi di ottimizzazione, quali il metodo *Newton,* il metodo *Quasi-Newton* e l'algoritmo *Levenberg-Marquardt* (tutti disponibili in Matlab) si rimanda a White (1992), Bishop (1995) e Ripley (1996).

2.3. *Un'applicazione*

Allo scopo di far comprendere la metodologia presentata nei paragrafi precedenti, a prescindere dalla formalizzazione matematica sottostante (formalizzazione che può essere ignorata dall'utente più inesperto e delegata a pacchetti statistici già esistenti) e focalizzando invece l'attenzione sulle fasi inerenti all'applicazione del MLP, presentiamo ora un esempio numerico. Consideriamo i dati relativi a 241 società di capitali del settore tessile, nella provincia di Prato. Volendo classificare le aziende in «sane» e «fallite», sulla base dei valori assunti da 25 indicatori di bilancio e del settore di appartenenza, cerchiamo di trovare la funzione $y=f(\mathbf{x})$ con y variabile dicotomica che

assume valore 1 per le aziende fallite e 0 per le sane, ed $\mathbf{x}$ vettore di 26 componenti (25 variabili continue ed il codice numerico della variabile qualitativa, relativa al comparto tessile di appartenenza). Essendo 11 i possibili settori, il MLP avrà (25+11)=36 neuroni nel primo livello (somma delle 25 variabili continue e delle 11 modalità della variabile qualitativa codificata in maniera disgiuntiva). Per ogni azienda, il valore degli ultimi 11 neuroni di *input* sarà pari ad 1 per il comparto di appartenenza e pari a zero per tutti gli altri dieci comparti tessili. Il livello di *output* della rete avrà un solo neurone. Per le osservazioni relative al *training set* (con classificazione «sana» o «fallita» conosciuta) il valore di tale nodo sarà posto pari a 0 o ad 1. Per quanto riguarda l'*output* effettivo della rete, essendo il codominio della funzione lineare di trasferimento dal secondo al terzo livello uguale all'asse reale, il nodo potrà assumere un qualsiasi valore continuo, positivo o negativo. Tale valore potrà automaticamente essere posto pari a 0, e quindi l'azienda classificata come «sana», se è minore di 0.5, e posto uguale ad 1, e quindi l'azienda classificata come «fallita», se è maggiore di 0.5. Consideriamo un *training set* composto dalle 241 aziende di cui si conoscono le coppie $\{y_i, \mathbf{x}_i\}$ i=1, 2, ..., 241. Per quanto riguarda l'architettura di rete, consideriamo le possibili opzioni proposte dal programma *Neural Connection* di *SPSS.* Scegliamo di normalizzare i vettori di *input*, per eliminare la variabilità dovuta al diverso ordine di grandezza degli indici. Scegliamo un solo livello nascosto, con 10 neuroni ed una funzione di trasferimento logistica («*sigmoid*»). Non normalizziamo la variabile *target* e determiniamo il valore dei pesi iniziali mediante l'estrazione casuale da una distribuzione uniforme nell'intervallo (−0.1, 0.1) (tali valori saranno quindi concentrati attorno allo zero). Scegliamo il *conjugate gradient* quale regola di apprendimento. Tale algoritmo non comporta la determinazione di altri parametri, al contrario dello *steepest descent* che implica, ad esempio, la scelta del *learning rate* e del momento. Impostiamo l'apprendimento per epoca e non per esempio. Diamo un numero massimo di iterazioni pari a 1500. Alla fine dell'addestramento le aziende classificate correttamente sono il 98%. Una volta addestrata la rete e determinati, quindi, i valori dei parametri della funzione $y=f(\mathbf{x})$, possiamo utilizzarla per classificare nuove aziende di cui si vuole conoscere la solidità finanziaria o individuare il possibile rischio di fallimento. Sottolineiamo come non sia necessario conoscere i valori dei parametri stimati che, per questa rete, sono uguali a $(37\times10 + 11\times1) = 381$ (considerando il *bias* nel primo e nel secondo

livello). Tali parametri, al contrario, per esempio, dell'intercetta e del coefficiente angolare nel modello di regressione semplice, che hanno una precisa interpretazione, non hanno significato in sé. In questo senso le reti neurali si dicono modelli non-parametrici, perché vi sono molti parametri che, singolarmente, non hanno un preciso significato, ma offrono una possibilità di interpretazione solo se considerati insieme, ed in base ai risultati della funzione $y=f(\mathbf{x})$ a cui sono associati. Quindi non ci interessa esplicitare la funzione. Sappiamo che essa raggiunge un buon adattamento sui dati che abbiamo presentato per addestrarla e che, in base ai 26 indicatori utilizzati, classifica le aziende in «sane» e «fallite» commettendo un errore molto piccolo, pari al 2%. Il modello di rete individuato potrà essere utilizzato da operatori economici quali, ad esempio, le banche, per classificare le aziende che chiedono un prestito e vedere la loro solidità finanziaria. Un'azienda classificata dalla rete come «fallita» sarà considerata a rischio o, comunque, simile ad altre che, precedentemente, hanno subito il fallimento. Le aziende individuate come «sane» presumibilmente mostreranno una buona solidità finanziaria ed un rischio di fallimento molto minore in tempi ravvicinati.

Le unità classificate in modo errato sono tutte aziende sane il cui *output* determinato dalla rete è superiore a 0.5. L'errore più grave dal punto di vista d'un operatore economico come la banca, ovvero il classificare non a rischio aziende che effettivamente lo sono, risulta invece pari a zero.

La percentuale di aziende classificate correttamente dopo 1500 iterazioni non varia se si cambiano i valori dei pesi iniziali o si seleziona, come algoritmo di apprendimento, lo *steepest descent* con i valori di *default* proposti dal programma. Se si aumenta, invece, il numero di nodi nascosti, migliora l'adattamento della rete. Passando, ad esempio, da 10 a 20 nodi, la percentuale di aziende classificate in maniera corretta raggiunge il 98.5%. Con un numero ancora maggiore di neuroni intermedi la rete raggiunge il 100%. Il conseguimento di un adattamento perfetto non è, però, lo scopo finale dell'addestramento, a causa dell'insorgere dell'*overfitting*. Il problema dell'*overfitting* verrà illustrato nel paragrafo 4.

3. *Le reti* radial basis functions (RBF)

Le reti *radial basis function* (RBF) sono una speciale classe di reti neurali artificiali con un solo livello nascosto e in cui la funzione di *output* è espressa come combinazione lineare di m funzioni non lineari, dette *basis functions*. La principale differenza rispetto al *multi- layer perceptron* risiede nella funzione di attivazione $a_j = h(\mathbf{x}, \mathbf{w}_j)$ dei neuroni del livello nascosto. Nel *multi-layer perceptron* a_j è la combinazione lineare dei valori di *input* e dei pesi delle connessioni. Nelle reti RBF a_j è funzione della distanza tra il vettore di *input* ed un vettore caratteristico del nodo j, detto anche centro o *reference vector*. Per una esposizione dettagliata sulle delle differenze fra le reti *radial basis functions* ed il *multi-layer perceptron* si rimanda a Morlini (2001). Per una ampia trattazione sulle reti RBF si rinvia a Morlini (2002a) e (2002b) ed alla bibliografia ivi citata. In termini formali:

$$a_j = \phi(\|\mathbf{x}-\mu_j\|) \tag{12.25}$$

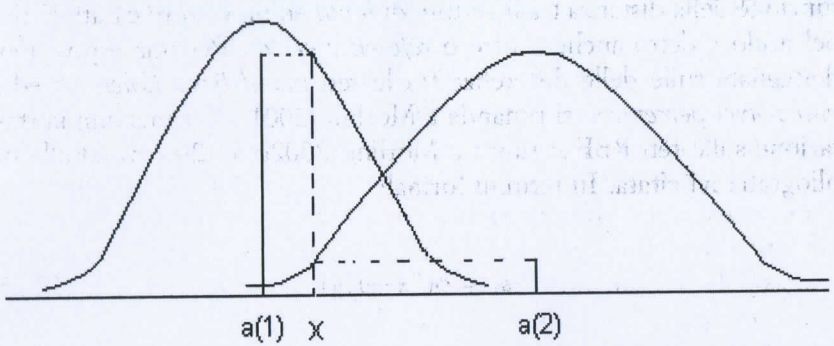
dove ϕ è una funzione simmetrica non lineare e μ_j, detto vettore caratteristico, è un vettore p-dimensionale le cui componenti sono le coordinate nello spazio di *input* che determinano il centro della funzione ϕ. Tale funzione può assumere diverse tipologie. In questa sede considereremo unicamente l'espressione più comune di ϕ, ovvero la funzione Gaussiana, che raggiunge il suo valore massimo, pari ad uno, in corrispondenza di $\mathbf{x}=\mu_j$, decresce al crescere della distanza tra $\mathbf{x}$ e μ_j, molto rapidamente all'inizio e poi asintoticamente verso lo 0, ed ha un'ampiezza determinata da un ulteriore parametro, σ_j. Se la distanza scelta per $\|\mathbf{x} - \mu_j\|$ è la distanza Euclidea, la formula precedente può essere riscritta come segue:

$$a_j = \exp\left(-\frac{\sum_{s=1}^{p}(x_s - \mu_{sj})^2}{2\sigma_j^2}\right) \tag{12.26}$$

Unità nascoste con questo tipo di attivazione hanno una caratteristica fondamentale: sono localizzate, ovvero assumono un valore significativamente diverso da zero solo per vettori $\mathbf{x}$ in un'area limitata dello spazio di *input*. La fig. 12.12 illustra il calcolo delle funzioni di attivazione di due nodi nascosti di una rete RBF con un solo valore in entrata. Lo spazio di *input* è rappresentato dall'asse orizzontale. La linea tratteggiata verticale mostra il valore dell'osservazione proposta alla rete. Le due linee continue mostrano i centri delle funzioni radiali e le loro altezze gli stati di attivazione dei rispettivi nodi nascosti (le altezze sono date dall'intersezione della linea tratteggiata verticale con

le due curve). Nell'esempio illustrato, l'unità 1 ha uno stato di attivazione elevato, l'unità 2 molto modesto.

FIG. 12.12. *Calcolo della funzione di attivazione in una rete RBF.*



I nodi del livello di *output* calcolano la combinazione lineare dei valori a_j che ricevono dal livello nascosto ed il generico *output* della rete y_k può essere così esplicitato:

$$y_k = \sum_{j=1}^{m} w_{kj} a_j + w_{k0} = \sum_{j=1}^{m} w_{kj} \exp\left(-\frac{\sum_{s=1}^{p}(x_s - \mu_{sj})^2}{2\sigma_j^2}\right) + w_{k0} \quad (12.27)$$

dove w_{k0} è una costante, detta *bias*, spesso omessa nelle applicazioni. L'*output* y_k può essere descritto come una somma ponderata in cui il peso w_{kj} rappresenta il valore desiderato per un'osservazione $\mathbf{x}$ esattamente uguale al centro $\boldsymbol{\mu}$: Lo stato di attivazione a_j, compreso nell'intervallo (0,1] indica il grado di vicinanza di $\mathbf{x}$ a $\boldsymbol{\mu}$, ed il nodo di *output* usa questo valore per ponderare il corrispondente parametro w_{kj}.

Come semplice esempio, consideriamo il problema logico «XOR» (date due asserzioni, che chiameremo asserzione «A» ed asserzione «B», l'unione di A e B tramite l'operatore logico XOR risulta vera se e solo se o A è vera e B è falsa o B è vera ed A è falsa). Consideriamo due nodi di *input*, rappresentanti, ad esempio, gli addendi della somma di due termini (interi e positivi), sia 1 il valore che essi assumono se il numero è dispari e 0 se è pari. Sia l'*output* desiderato uguale ad 1 se la somma è un numero dispari, uguale a zero in caso contrario. Le possibili alternative sono riportate nella tabella 12.2.

TAB. 12.2. *Funzione logica XOR.*

Primo input	secondo input	target
1	1	0
1	0	1
0	1	1
0	0	0

Tale problema può essere risolto esattamente da una rete RBF con due nodi di *input*, un nodo di *output*, quattro *basis functions*. Poniamo $\sigma_j = \sigma = 0.85 (j = 1,2,3,4)$ e $\mu_{11} = 1$, $\mu_{21} = 1$, $\mu_{12} = 1$, $\mu_{22} = 0$, $\mu_{13} = 0$, $\mu_{23} = 1$, $\mu_{14} = 0$, $\mu_{24} = 0$ (le modalità di stima dei parametri verranno illustrate nel prossimo paragrafo). Se le connessioni tra il secondo ed il terzo livello sono: $w_{11} = -1.778$, $w_{21} = 2.222$, $w_{31} = 2.222$, $w_{41} = -1.778$, (senza il *bias*) allora per il primo esempio si ha:

$$a_1 = \exp\left(-\frac{(1-1)^2 + (1-1)^2}{2(0.7225)}\right) = \exp(0) = 1$$

$$a_2 = \exp\left(-\frac{(1-1)^2 + (1-0)^2}{2(0.7225)}\right) = \exp(-0.692) = 0.5$$

$$a_3 = \exp\left(-\frac{(1-0)^2 + (1-1)^2}{2(0.7225)}\right) = \exp(-0.692) = 0.5$$

$$a_4 = \exp\left(-\frac{(1-0)^2 + (1-0)^2}{2(0.7225)}\right) = \exp(-1.384) = 0.25$$

da cui:

$y(1,1) = -1.778 \times 1 + 2.222 \times 0.5 + 2.222 \times 0.5 - 1.778 \times 0.25 \doteq 0.$

Per il secondo esempio:

$$a_1 = \exp\left(-\frac{(1-1)^2 + (0-1)^2}{2(0.7225)}\right) = \exp(-0.629) = 0.5$$

$$a_2 = \exp\left(-\frac{(1-1)^2 + (0-0)^2}{2(0.7225)}\right) = \exp(0) = 1$$

$$a_3 = \exp\left(-\frac{(1-0)^2+(0-1)^2}{2(0.7225)}\right) = \exp(-1.384) = 0.25$$

$$a_4 = \exp\left(-\frac{(1-0)^2+(0-0)^2}{2(0.7225)}\right) = \exp(-0.269) = 0.5$$

da cui:
$y(1,0) = -1.778 \times 0.5 + 2.222 \times 1 + 2.222 \times 0.25 - 1.778 \times 0.5 = 1.$

Per il terzo esempio:

$$a_1 = \exp\left(-\frac{(0-1)^2+(1-1)^2}{2(0.7225)}\right) = \exp(-0.629) = 0.5$$

$$a_2 = \exp\left(-\frac{(0-1)^2+(1-0)^2}{2(0.7225)}\right) = \exp(-1.384) = 0.25$$

$$a_3 = \exp\left(-\frac{(0-0)^2+(1-1)^2}{2(0.7225)}\right) = \exp(0) = 1$$

$$a_4 = \exp\left(-\frac{(0-0)^2+(1-0)^2}{2(0.7225)}\right) = \exp(-0.629) = 0.5$$

da cui:
$y(0,1) = -1.778 \times 0.5 + 2.222 \times 0.25 + 2.222 \times 1 - 1.778 \times 0.5 = 1.$

Per il quarto esempio:

$$a_1 = \exp\left(-\frac{(0-1)^2+(0-1)^2}{2(0.7225)}\right) = \exp(-1.384) = 0.25$$

$$a_2 = \exp\left(-\frac{(0-1)^2+(0-0)^2}{2(0.7225)}\right) = \exp(-0.629) = 0.5$$

$$a_3 = \exp\left(-\frac{(0-0)^2+(0-1)^2}{2(0.7225)}\right) = \exp(-0.629) = 0.5$$

$$a_4 = \exp\left(-\frac{(0-0)^2+(0-0)^2}{2(0.7225)}\right) = \exp(0) = 1$$

da cui:
$y(0,0) = -1.778 \times 0.25 + 2.222 \times 0.5 + 2.222 \times 0.5 - 1.778 \times 1 = 0.$

3.1. *Le modalità di apprendimento*

L'apprendimento, ovvero la ricerca dei valori ottimali dei parametri, è diviso in due fasi: la prima per stimare i parametri σ_j e μ_j delle unità nascoste, la seconda per stimare i parametri w_{kj} del secondo livello. I parametri σ_j e μ_j controllano, rispettivamente, la larghezza e il centro delle funzioni a campana calcolate dalle unità nascoste. I valori di tali parametri devono essere stimati in modo che le unità nascoste «ricoprano» la parte dello spazio di *input* in cui sono le osservazioni, con una densità proporzionale alla densità delle osservazioni in ogni area e in modo che le larghezze forniscano un adeguato livello di sovrapposizione. I parametri σ_j e μ_j dipendono unicamente dalla distribuzione delle variabili di *input*, e non dalla relazione fra *input* ed *output*. Per l'apprendimento si usano, quindi, tecniche non supervisionate, che richiedono la conoscenza delle sole variabili di *input* ed ignorano i valori di *target* e per questo molto veloci dal punto di vista computazionale. L'apprendimento più semplice consiste nel porre tante unità nascoste m quante sono le osservazioni nel *training set* e considerare come centri i vettori di *input*. In termini formali, ponendo $m = n$:

$$\mu_j = \mathbf{x}_i (j = 1, \ldots, n; i = 1, \ldots, n) \qquad (12.28)$$

In alternativa, quando il numero di osservazioni n è particolarmente elevato, possono essere considerati come centri un sottoinsieme di m vettori di *input* ($m < n$), scelti casualmente o tramite opportune tecniche di selezione (fra queste, le più conosciute sono la *forward selection* (Orr (1993)) e i minimi quadrati ortogonali (Chen *et al.* (1991)). Un approccio diverso, non contemplato nel pacchetto *Neural Connection*, è quello di utilizzare algoritmi di *clustering* (come, ad esempio, il metodo non gerarchico delle k medie) e porre i parametri μ_j uguali ai centroidi individuati da tali algoritmi (per un'applicazione si veda Morlini (1999)).

Per quanto riguarda i parametri σ_j, l'approccio più semplice è quello di scegliere un valore comune σ per tutte le unità dato, ad esempio, dalla media delle distanze fra ciascun centro ed il più vicino (oppure dalla media delle distanze fra tutte le coppie di centri). Se si ritiene, invece, che le unità debbano avere un differente grado di *overlapping* e che i valori di σ_j debbano essere diversi, l'approccio più seguito è quello di considerare la media delle distanze da ogni centro e gli z centri più vicini (per un determinato valore di z). Tale opportunità non è con-

templata nel pacchetto *Neural Connection*, che limita la scelta ad un solo valore σ comune a tutte le *basis functions*.

L'apprendimento dei parametri w_{kj} del secondo livello richiede la conoscenza dei valori di *target*. Tale apprendimento, pur essendo supervisionato, risulta molto più veloce rispetto a quello del *multi-layer perceptron*. Infatti, una volta determinati i parametri delle unità nascoste, il problema di minimizzazione dell'errore quadratico può essere ricondotto a quello della stima dei pesi in una regressione lineare e quindi alla risoluzione di un sistema di m equazioni lineari in m incognite. La funzione di errore è, in questo caso, una funzione convessa, con un unico minimo globale.

Da un punto di vista teorico, risulta ovviamente possibile stimare congiuntamente i parametri μ_j, σ_j e w_{kj} tramite le tecniche iterative illustrate nel capitolo precedente. Tuttavia, utilizzando tali tecniche, si perderebbe la principale caratteristica delle reti RBF, data dalla velocità di apprendimento e dovuta alla divisione di questo problema, particolarmente difficile nel suo complesso, in due parti semplici.

3.2. *Un'applicazione*

Riprendiamo in considerazione il data set sugli *home theater* utilizzato nell'applicazione di vari altri metodi illustrati in questo volume (vedi tabella 1.1). L'obiettivo è ora quello di stimare, per un nuovo modello da immettere sul mercato, un prezzo concorrenziale. A tal fine supponiamo che il prezzo di acquisto debba dipendere dalle caratteristiche tecniche del prodotto quali la potenza, il numero di casse, la presenza o meno di *dvd recorder*, di tecnologia *wireless*, di *dvx* e di *radio data system*. Per queste ultime 4 variabili la codificazione sarà 1 se la caratteristica tecnica è presente, 0 se è assente. Le variabili possono essere quindi trattate come variabili quantitative. Cerchiamo di stimare la funzione $y=f(\mathbf{x})$; con y = prezzo ed $\mathbf{x}$ = vettore di 6 componenti. Il nostro *training set* è composto da 25 osservazioni $\{y_i, \mathbf{x}_i\}$ $i = 1, 2, \ldots, 25$ relative ai 25 modelli di *home theater* per i quali si conoscono il prezzo e le caratteristiche tecniche. La rete RBF avrà 6 neuroni nel primo livello ed un neurone nel terzo livello, essendo $\mathbf{x}_i$ il vettore di *input* ed y_i il valore di *target*. Per quanto riguarda la configurazione del modello di rete, consideriamo le possibili opzioni proposte dal programma *Neural Connection*. Scegliamo di normalizzare i vettori di *input*, per trasformare anche i valori della potenza e del numero di casse in nuovi valori compresi nell'intervallo [0,1]. La normalizzazione o la standardizzazione sono particolarmente importanti per queste reti che si basano sul calcolo di distanze. Per quanto riguarda il livello nascosto, non determiniamo a priori il numero di nodi, ma deleghiamo all'algoritmo tale scelta. Con l'attivazione dell'opzione «*automatic node generation*» il *software* effettua diverse prove con un differente numero di nodi e determina il numero ottimale in base al minimo errore quadratico raggiunto (le prove iniziano con un solo neurone nascosto e terminano quando, al crescere dei nodi, non corrisponde più una diminuzione consistente nell'errore). Scegliamo la distanza Euclidea e la funzione Gaussiana (nel pacchetto sono disponibili anche altre distanze, come quella di *Manhattan*, ed altre possibili *basis functions*, come la funzione quadratica e le *splines*). Immettiamo un valore di σ costante in ogni *basis function*, pari a 0.5. Determiniamo il valore dei centri estraendo casualmente, con un seme di estrazione pari a 12 (lo stesso seme comporta sempre l'estrazione degli stessi elementi da un determinato campione), un ugual numero di vettori di *input* dal *training set*. Alternativamente, i centri potevano essere determinati in maniera del tutto casuale, ma in questo caso i valori avrebbero potuto essere anche molto diversi da quelli assunti dai 6 indicatori considerati. Il numero ottimale di centri risulta pari a 3. L'addestramento è velocissimo, pari ad una frazione di secondo. Il *training set* è, infatti, di limitate dimensioni, se si considera che nelle applicazioni le osservazioni proposte alla rete sono, di solito, nell'ordine delle migliaia e non delle decine. Una volta addestrata la rete e determinati, quindi, i valori dei parametri della funzione $y=f(\mathbf{x})$, possiamo utilizzarla per stimare un prezzo concorrenziale di mercato per un nuovo prodotto. Ad esempio, un ipotetico nuovo modello con una potenza pari a 500, 6 casse, senza il *dvd recorder*, senza la tecnologia *wireless* ed il *dvx* ma con il *radio data system*, avrebbe un prezzo stimato pari a 350 euro, essendo f (500, 6, 0, 0, 0, 1)=350. Per essere competitiva rispetto ai modelli con analoghe caratteristiche già presenti sul mercato, quindi, l'azienda produttrice dovrebbe immettere sul mercato il prodotto con un prezzo inferiore ai 350 euro stimati mediante la rete.

Proviamo ad addestrare la rete con diversi valori di σ e con differenti centri (cambiando il seme) per vedere la robustezza del modello neurale selezionato rispetto a questi parametri. I centri influiscono sui risultati in modo trascurabile. Il valore di σ influisce, invece, in maniera determinante, soprattutto oltre una certa soglia, ovvero quando diventa «troppo piccolo» o «troppo grande». Nel prossimo paragrafo cercheremo di chiarire queste due asserzioni.

4. Overfitting *e generalizzazione*

Un problema particolarmente avvertito nelle applicazioni delle reti neurali come il *multi-layer perceptron* e le reti RBF è quello dell'*overfitting*. Con questo nome ci si riferisce alla tendenza del modello a specializzarsi troppo sui dati utilizzati nell'addestramento. In situazioni di *overfitting* la rete interpola non solo la porzione del fenomeno riconducibile al legame strutturale tra le variabili di *input* e quelle di *output*, ma anche quella imputabile al verificarsi della componente erratica. La conseguenza è di ottenere misure di adattamento molto buone sul *training set*, ma decisamente peggiori se calcolate su dati diversi da quelli utilizzati nella fase di apprendimento. L'*overfitting* risulta in diretta competizione con la capacità di generalizzazione della rete, intendendo con questo termine proprio l'attitudine ad offrire buone *performances* su dati diversi dal *training set*.

Nel *multi-layer perceptron* la tendenza all'*overfitting* è conseguenza di un numero elevato di nodi nel livello nascosto che può determinare un eccesso di parametri liberi rispetto a quanto richiesto dal fenomeno in esame. Per contrastare il fenomeno in esame appare dunque appropriato considerare dei vincoli nella determinazione dei parametri **w**. Il modo più diretto consiste nel porre un fattore di penalizzazione all'errore e minimizzare la seguente funzione obiettivo (detta anche funzione di costo):

$$C = E + \lambda\Omega\ (\mathbf{w}) \qquad (12.29)$$

dove λ è uno scalare di controllo e Ω è una funzione positiva sui pesi **w**, detta funzione di regolarizzazione. La forma più semplice per Ω è data da:

$$\Omega = (1\backslash 2)\mathbf{w}'\mathbf{w} \qquad (12.30)$$

che porta all'algoritmo di apprendimento noto come *weight decay* (nel pacchetto *Neural Connection* tale algoritmo è usato nelle reti denominate reti bayesiane). Il *weight decay* possiede una precisa interpretazione se considerato dal punto di vista bayesiano. In questo contesto ci limitiamo a chiarire il senso di tale interpretazione, senza entrare nei dettagli delle dimostrazioni e senza descrivere tutta la teoria sottostante all'interpretazione bayesiana del multi-layer perceptron e i conseguenti metodi di stima dei parametri. Per rassegne ed applicazioni dei metodi bayesiani all'ambito neurale si rimanda a Spiegelharter *et al.* (1993) e Bishop (1995). Ipotizzando che l'errore commesso dalla rete addestrata si distribuisca secondo una legge di tipo multinormale con matrice di dispersione sferica e con vettore delle medie nullo, allora aggiungere all'errore quadratico il fattore di penalizzazione di tipo *weight decay* equivale a specificare una distribuzione iniziale sui pesi anch'essa di forma multinormale. La distribuzione iniziale rifletterà la convinzione che tutti i pesi presentino la stessa incertezza ed i valori in partenza più probabili siano da ritenersi concentrati intorno allo zero. Minimizzare la funzione di costo significa trovare la moda della densità a posteriori definita sul vettore dei pesi.

Nelle reti RBF la capacità di generalizzazione è controllata non solo dal numero di nodi nascosti, ma anche dal valore dei parametri σ_j. Si supponga, ad esempio, di considerare come centri tutte le osservazioni presenti nel *training set* e di porre i parametri σ_j piccoli in modo tale da avere un valore di attivazione dell'unità nascosta prossimo allo zero per tutti i vettori di *input*, ad eccezione di quello che costituisce il centro dell'unità. La funzione calcolata dalla rete passerà esattamente attraverso ogni elemento del *training set*, e l'errore calcolato su questo sarà quindi pari a zero. Se i dati del *training set* sono affetti da rumore, come quasi sempre accade nei problemi reali, questa funzione, molto complessa e particolarmente ondulata, produrrà un'elevata percentuale di errore su nuovi esempi. Per determinare dei vincoli sui parametri σ_j non vi è una metodologia analoga al *weight decay*. Nelle applicazioni la scelta di questi parametri avviene in maniera euristica, considerando diversi valori e controllando il grado di generalizzazione della rete attraverso tecniche di selezione del modello, illustrate nel prossimo paragrafo.

4.1. *Tecniche di selezione del modello*

Tali tecniche sono volte a quantificare l'errore di una rete neurale su osservazioni diverse da quelle usate nella stima dei parametri. Possono essere individuate due classi di metodi per la selezione di un modello neurale: metodi che si basano su un campione diverso dal *training set* e metodi che si basano su indici calcolati sull'insieme delle osservazioni su cui viene effettuata la stima dei parametri. La prima classe richiede la disponibilità di un numero elevato di osservazioni. Tipicamente, le osservazioni vengono divise in due parti: un insieme

di apprendimento (il *training set*) e un insieme su cui stimare l'errore per la scelta del modello (il *validation set*). Alcune applicazioni prevedono anche un terzo gruppo, detto *test set*, su cui misurare l'errore di previsione (sottolineiamo come tali definizioni non siano univoche ed alcuni autori usino indifferentemente i termini *validation set* e *test set*).

In alternativa, la tecnica di *cross validation* (Stone (1974), Efron (1983), Efron e Tibshirani (1993)) implica la divisione delle osservazioni in k sottoinsiemi di uguale numerosità. La rete viene addestrata k volte, ogni volta tralasciando uno dei k sottogruppi e calcolando l'errore su questo. L'errore finale di generalizzazione è dato dalla media dei k errori ottenuti nelle diverse prove. Nel caso particolare in cui $k=n$, la rete viene addestrata n volte, ogni volta tralasciando una singola osservazione, e l'errore di *cross validation* è anche detto *leave- one-out*.

Nell'esempio relativo alla classificazione di aziende in «sane» e «fallite» (paragrafo 2.3), possiamo dividere il campione di 241 unità in due sub-campioni: un *training set* di 120 aziende su cui addestrare la rete ed un *validation set* di 121 osservazioni, su cui stimare l'errore di previsione. Dopo 1500 iterazioni il MLP raggiunge una percentuale di aziende classificate correttamente pari al 97% nel *training set* e al 55% nel *validation set*. La rete mostra, quindi, un buon adattamento sui dati utilizzati per l'addestramento ma una ridotta capacità di previsione. Se l'addestramento viene interrotto dopo 1000 iterazioni, la percentuale di aziende corrette nel *training set* diminuisce e passa al 95%, mentre quella relativa al *validation set* raggiunge l'89.2%. Il modello risulta più affidabile per la classificazione di aziende «nuove» e diverse da quelle utilizzate per l'addestramento. I risultati che si ottengono nel *validation set* sono elencati nella seguente tabella a doppia entrata

	Aziende sane	Aziende fallite
Aziende classificate sane	95 (78.5%)	4 (3.3%)
Aziende classificate fallite	9 (7.5%)	13 (10.7%)

Nell'esempio relativo agli "home theater" (paragrafo 3.2), possiamo dividere il campione delle osservazioni in un *training set* di 20 esempi ed un *validation set* formato dai 5 rimanenti. In questo caso, il numero ottimale di neuroni nascosti e, quindi, il modello di rete finale

da utilizzare per la stima, viene determinato in base all'errore sul *validation set*. Nel caso precedente, addestrando e controllando l'errore della rete sulle stesse osservazioni, il numero era risultato pari a 3. Minimizzando l'errore sul *validation set*, il numero passa da 3 a 2. Dunque, la maggior flessibilità data dal neurone aggiuntivo può ragionevolmente essere imputata al fenomeno dell'*overfitting* e non tanto al legame strutturale esistente tra i 6 indicatori utilizzati ed il prezzo. Essendo poco elevata la numerosità sia del *training* sia del *validation set*, più convenientemente avremmo potuto utilizzare la tecnica di *cross validation* formando, ad esempio, cinque gruppi di 5 modelli e identificando tali gruppi con i numeri 1, 2, 3, 4, 5. La rete avrebbe dovuto essere addestrata cinque volte. La prima volta con i modelli dei gruppi 1, 2, 3, 4, la seconda con quelli dei gruppi 1, 2, 3, 5, la terza con quelli dei gruppi 1, 2, 4, 5, la quarta con quelli dei gruppi 1, 3, 4, 5, e, infine, la quinta, con quelli dei gruppi, 2, 3, 4, 5. La media degli errori calcolati su 5, 4, 3, 2, 1, rispettivamente, avrebbe costituito l'errore di previsione. Ovviamente il costo computazionale sarebbe stato maggiormente elevato.

La seconda classe di metodi per la selezione di un modello neurale richiede un numero inferiore di osservazioni, poiché tutte intervengono congiuntamente sia nella stima dei parametri della rete, sia nella stima dell'errore di previsione. I criteri di selezione del modello appartenenti a tale classe si fondano su indici che, sottolineiamo, pur essendo calcolati sullo stesso campione utilizzato per la determinazione dei parametri, danno una stima dell'errore «al di fuori» del campione. Tali indici si basano sul numero effettivo di parametri ν che, tipicamente, è molto inferiore al numero di parametri stimati (a tal proposito si vedano Ingrassia e Morlini (2002) e (2005)). Per il calcolo di ν e per una più completa esposizione di questo tema si rimanda inoltre a Moody (1992) e MacKay (1992). In questa sede ci limitiamo ad elencare alcuni degli indici che possono essere utilizzati. Essi sono la *stima corretta della varianza* (UEV: *unbiased estimate of variance*):

$$\text{UEV} = \text{MSE}\left(\frac{n}{n-\nu}\right) \tag{12.31}$$

l'indice AIC (*Akaike's Information Criterion*):

$$\text{AIC} = \log \text{MSE} + 2\nu\left(\frac{1}{n}\right) \tag{12.32}$$

l'indice BIC (*Schwartz's Bayesian Information Criterion*):

$$\text{BIC} = \log \text{MSE} + \nu\left(\frac{\log n}{n}\right) \qquad (12.33)$$

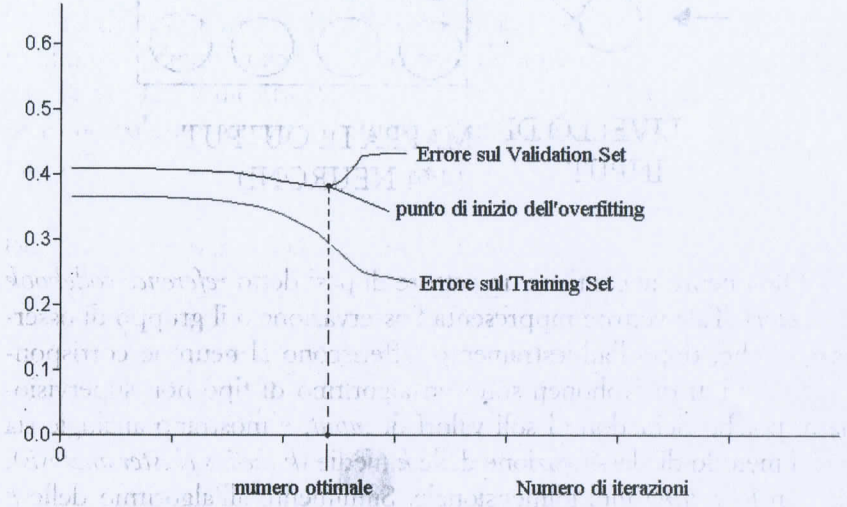
dove MSE (*mean squared error*) è la radice (presa con segno positivo) dell'errore quadratico medio sul *training set*:

$$\text{MSE} = \sqrt{\frac{1}{nc}\sum_{i=1}^{n}\sum_{k=1}^{c}(y_{ik} - t_{ik})^2} \qquad (12.34)$$

La funzione calcolata per la selezione del modello, sia essa l'errore di *cross validation*, l'errore calcolato sul *validation set*, la stima corretta della varianza, l'AIC o il BIC, ha un andamento diverso rispetto all'errore calcolato sul *training set*. Tipicamente, aumentando il numero di nodi nascosti, sia nel *multi-layer perceptron* sia nelle reti RBF, il secondo decresce mentre la prima decresce sino ad un certo livello e poi inizia ad aumentare. È questo il punto in cui interviene l'*overfitting*. Il numero ottimale di nodi nascosti è quello associato al minimo della funzione calcolata per la selezione del modello, prima dell'*overfitting*. In maniera analoga, i due tipi di errore possono essere messi in relazione con il valore stimato di σ (supposto uguale in tutte le *basis functions*) in una rete RBF, oppure con il vettore dei pesi $\mathbf{w}(t)$ relativo ad ogni epoca di apprendimento t, nel *multi-layer perceptron*. Anche il tempo di addestramento, infatti, influisce sul livello di adattamento dei pesi e quindi sulla capacità di generalizzazione della rete. La fig. 12.13 illustra gli andamenti degli errori del *multi-layer perceptron* sul *training* e sul *validation set*, al crescere del numero di iterazioni, nell'esempio relativo alla classificazione delle aziende.

In generale, nella applicazioni gli errori mostrano andamenti simili a quelli presentati nella figura. Il grafico viene ottenuto calcolando, ad ogni epoca di apprendimento, non solo l'errore che viene minimizzato per determinare i parametri $\mathbf{w}(t)$, ma anche quello per la selezione del modello. La cosiddetta regola dell'*arresto precoce* (o *stopping training*) prevede di interrompere l'addestramento (anche prima di aver raggiunto la convergenza) nel momento in cui interviene l'*overfitting*, ovvero quando il secondo tipo di errore inizia a crescere, e scegliere come soluzione ottimale quel vettore di pesi $\mathbf{w}(t)$ a cui corrisponde il minimo di tale errore. Se quest'ultimo non aumenta mai, significa che la rete non ha abbastanza gradi di libertà per raggiungere una situazione di *overfitting*, e probabilmente non ha nemmeno una sufficiente flessibilità per approssimare la funzione *input-output*. Da cui l'opportunità di addestrare di nuovo la rete con un maggior numero di parametri liberi, ovvero con un maggior numero di nodi nascosti.

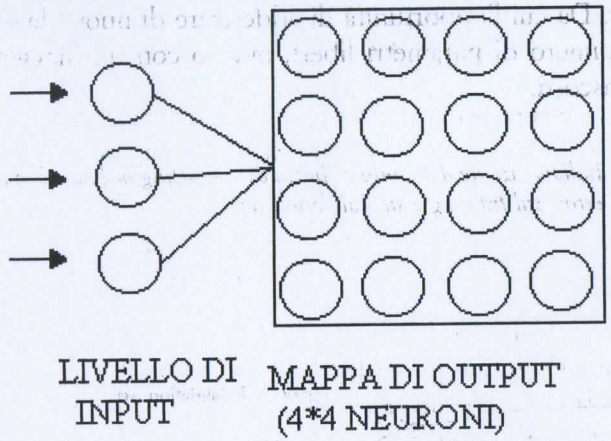
FIG. 12.13. *Individuazione del punto di inizio di overfitting in base al confronto fra gli errori sul training e sul validation set*



5. *Le reti di Kohonen* (Self-Organising-Maps)

L'obiettivo di fondo delle reti di Kohonen, dette anche *self-organising maps*, consiste nel fornire, per insiemi di dati in spazi di dimensione elevata, una rappresentazione sintetica in uno spazio, discreto, di dimensione inferiore. Tale spazio è costituito da un insieme di neuroni, disposti non su un livello, ma in una mappa quadrata o rettangolare (la topologia della mappa è solo quadrata nel pacchetto *Neural Connection* di SPSS), come mostra la fig. 12.14. Per eventuali approfondimenti su queste reti si rimanda a Kokonen (1997). In questo paragrafo ci limiteremo a descrivere brevemente il loro funzionamento e ad illustrare le problematiche statistiche che possono contribuire a risolvere.

FIG. 12.14. *Architettura della rete di Kohonen*



Ogni neurone contiene un vettore di pesi detto *reference codebook* (o *vector*). Tale vettore rappresenta l'osservazione o il gruppo di osservazioni che, dopo l'addestramento, afferiscono al neurone corrispondente. Le reti di Kohonen sono un algoritmo di tipo non supervisionato, poiché richiedono i soli valori di *input*, e mostrano analogie sia con il metodo di classificazione delle *k* medie (*k-means cluster analysis*), sia con lo *scaling* multidimensionale. Similmente all'algoritmo delle *k* medie, i dati vengono ordinati in una serie di gruppi (i neuroni) secondo le relazioni che la rete scopre tra i dati stessi, sulla base della minima distanza tra i vettori dei dati e i *reference codebooks* (assimilabili ai centroidi della *cluster analysis*). Come nello *scaling* multidimensionale, i *reference codebooks* sono spazialmente correlati e la distribuzione dei dati nel sotto-spazio cerca di riprodurre la struttura topologica dello spazio originario, in modo che gruppi vicini abbiano *reference vectors* simili. Al pari di queste due metodologie statistiche, le reti di Kohonen possono essere usate per l'individuazione di *outliers* multivariati (vedi Morlini, 2004).

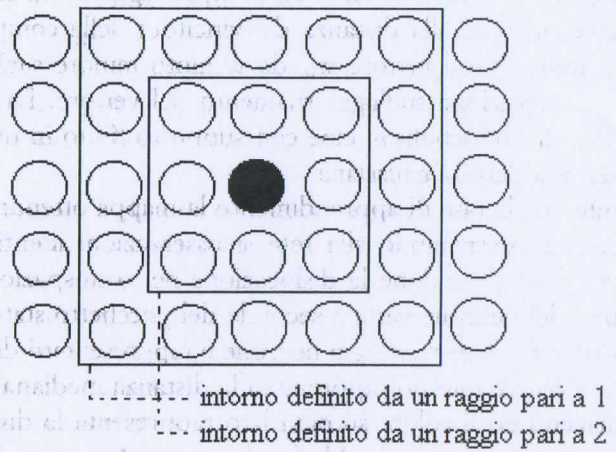
L'insieme dei pesi $\mathbf{w}_{rc}$ associati ad ogni neurone *rc* della mappa (dove *r* indica la riga e *c* la colonna) è generalmente inizializzato utilizzando un generatore di numeri casuali. Ogni volta che il vettore $\mathbf{x}_i$ $i = 1, \ldots, n$ viene proposto alla rete, il neurone associato al vettore $\tilde{\mathbf{w}}$ che mostra la minor distanza con $\mathbf{x}_i$ viene attivato, rivelando la posizione del dato nella mappa. Successivamente, non solo il vettore $\tilde{\mathbf{w}}$,

il cosiddetto «vincitore della competizione», ma anche alcuni neuroni vicini, vengono modificati secondo la regola di apprendimento:

$$\mathbf{w}_{rc}(t+1) = \mathbf{w}_{rc}(t) + ||\mathbf{x}_i - \mathbf{w}_{rc}(t)||\alpha(t) \qquad (12.35)$$

dove *t* indica il numero dell'iterazione (quante volte tutte le osservazioni *n* del *training set* sono state già proposte alla rete) ed α, nota come *learning rate*, è una costante compresa tra zero ed uno, che determina l'entità della modifica sul vettore. Il numero di neuroni che vengono modificati è determinato da un intorno definito da un raggio. Ad esempio, la fig. 12.15 mostra gli intorni del vincitore della competizione (il neurone colorato di nero) definiti da un raggio pari ad 1 e un raggio pari a 2.

FIG. 12.15. *Esempi di intorni in una rete di Kohonen.*



Durante il processo di apprendimento le osservazioni sono presentate alla rete più volte, sino alla convergenza o ad un determinato errore di codifica vettoriale (dato dalla media delle distanze tra le osservazioni associate ad ogni neurone ed i relativi *reference vectors*). La costante α e la definizione di intorno possono cambiare durante la fase di apprendimento. Tipicamente, sia il valore di α sia il raggio che determina l'ampiezza dell'intorno decrescono all'aumentare del numero delle iterazioni. L'effetto di questo processo è che all'inizio dell'apprendimento

la nuova informazione provoca aggiornamenti di entità più ampia e di carattere meno locale, per imprimere una certa struttura topologica alla mappa (di solito il raggio è definito in modo che l'intorno includa la maggior parte dei neuroni della mappa). Le ultime iterazioni, in cui il *learning rate* viene scelto prossimo allo zero ed il raggio vicino all'unità, consentono invece di raffinare la precisione della rappresentazione e dare potere di classificazione alla rete.

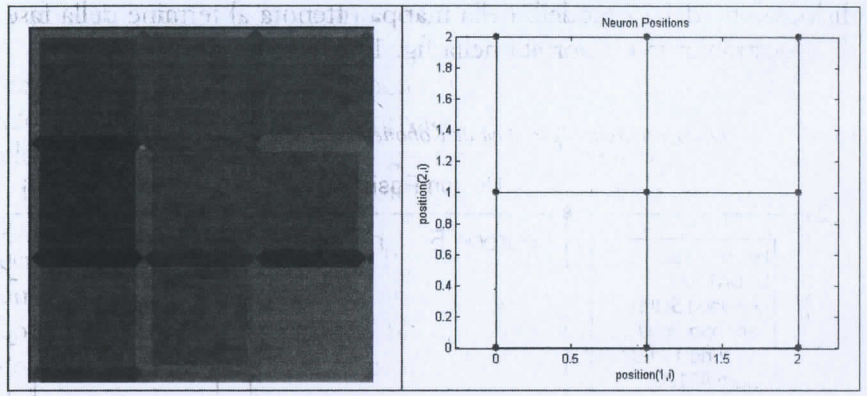
Una regola di apprendimento più elaborata, non presente nel pacchetto *Neural Connection* ma disponibile in molti altri programmi (come ad esempio in Matlab), è la seguente:

$$\mathbf{w}_{rc}(t+1) = \mathbf{w}_{rc}(t) + h_{rc}(||\mathbf{w}_{rc} - \tilde{\mathbf{w}}||) \times ||\mathbf{x}_i - \mathbf{w}_{rc}(t)||\alpha(t) \quad (12.36)$$

dove h_{rc} (il *neighborhood kernel*) è una funzione, decrescente, della distanza tra il neurone $\mathbf{w}_{rc}$ ed il vettore $\tilde{\mathbf{w}}$. Tale funzione consente di aggiornare i pesi definiti dall'intorno non in modo uguale ma, come sembra più logico, in base alla distanza dal vincitore della competizione. Quanto più lontano è il vettore $\mathbf{w}_{rc}$ da $\tilde{\mathbf{w}}$ tanto minore sarà l'effetto della nuova osservazione sull'aggiornamento del vettore. La funzione h_{rc} può essere di tipo «bolla», cioè con supporto finito in un intorno del neurone, o la densità gaussiana.

Al termine della fase di apprendimento la mappa ottenuta può essere interpretata presentando alla rete le osservazioni identificate da un'etichetta ed analizzandone la dislocazione nel sottospazio. La rappresentazione del neurone varia a seconda del pacchetto statistico utilizzato. In *Neural Connection* ogni neurone è rappresentato da un quadrato in cui il colore interno rappresenta la distanza mediana da tutti i neuroni adiacenti ed il colore su ogni lato rappresenta la distanza dal singolo neurone confinante. In Matlab la rappresentazione è data da un punto in una griglia che mostra le connessioni con i neuroni adiacenti (si veda, per il confronto fra i due tipi di rappresentazione, la fig. 12.16). Il risultato finale della rete dipende dalla distribuzione iniziale dei pesi, dalla metrica usata e dalla scelta dei parametri (si veda, ad esempio, Morlini, 1998).

FIG. 12.16. *Rappresentazione di una mappa di Kohonen 3×3 in Neural Connection (a sinistra) e Matlab (a destra)*



Esempio 1. Consideriamo, per chiarire quanto esposto sino ad ora, l'esempio proposto nella sezione 3.2. Il problema, in questa sede, è quello di classificare i 25 modelli di "home theater" in base ai valori assunti dalle 7 variabili (compreso il prezzo) e vedere il grado di somiglianza e diversità fra ogni modello. Scegliamo, per l'applicazione, una mappa di 3×2 neuroni. Visto le opzioni limitate del pacchetto *Neural Connection* relativamente alla scelta della topologia della mappa, utilizziamo Matlab. Il numero di neuroni e quindi di gruppi, deve essere rapportato al numero di osservazioni da classificare e deve essere deciso tenendo presente che:

— Con un numero limitato di neuroni, la mappa può non essere in grado di apprendere i *clusters* contenuti nei dati.

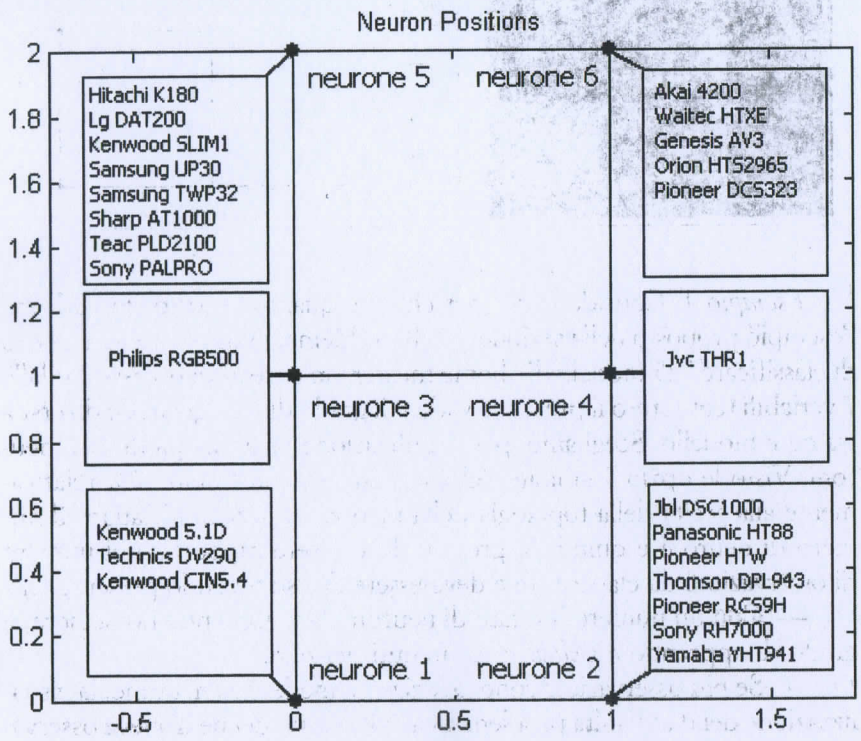
— Se più osservazioni sono associate al medesimo neurone, la visualizzazione dei dati risulta problematica (all'interno del neurone le osservazioni si distribuiscono in modo casuale e non è quindi possibile determinare il grado di somiglianza fra le unità in base alla distanza nella mappa).

— Con un numero considerevole di neuroni, molti di questi possono risultare vuoti.

Scegliamo una topologia di tipo rettangolare, una funzione h_{rc} gaussiana, un'inizializzazione dei pesi (6 vettori a 7 dimensioni) con estrazione casuale dai 25 vettori relativi ai modelli da classificare. Consideriamo la distanza Euclidea e normalizziamo le variabili per eliminare il differente campo di variazione. Per la prima fase di addestramento scegliamo un *learning rate* pari a 0.9, decrescente, in modo lineare, al crescere del numero delle iterazioni, sino a 0.02, un raggio iniziale pari a 3 ed uno finale pari a 1, un numero di iterazioni uguale a

1000. Per la seconda fase scegliamo, invece, un numero di iterazioni pari a 10000, un valore di α uguale a 0.02, ed un raggio pari a 1. La dislocazione dei 25 modelli nella mappa ottenuta al termine della fase di addestramento è riportata nella fig. 12.17.

FIG. 12.17. *Dislocazione nella mappa di Kohonen dei modelli di «home theater»*



La mappa evidenzia due aree: la prima, nella parte inferiore, relativamente omogenea al suo interno, formata dai neuroni 1 e 2 e la seconda, nella parte superiore, maggiormente disomogenea, formata dai neuroni 5 e 6. I neuroni 3 e 4 (ai quali afferisce una sola osservazione) fanno da collegamento fra queste due zone. Nella parte inferiore sono dislocati i modelli che, a fronte di un prezzo superiore alla media, mostrano anche una potenza ed un numero di casse superiore alla media. Nella parte superiore sono invece presenti modelli con prezzo e caratteristiche tecniche inferiori. Queste due aree identificano due diversi segmenti di mercato. Per meglio identificare le caratteristiche che contraddistinguono queste due zone della mappa possiamo analizzare i

centroidi (i vettori contenenti i le medie delle osservazioni afferenti al gruppo) o i *reference vectors* di ogni neurone. Questi due vettori saranno simili, ma non necessariamente coincidenti.

Il centroide del gruppo 1 è caratterizzato da valori elevati nel prezzo, nella potenza e nel numero di casse (superiori alla media), da valori prossimi ad uno nelle altre variabili e pari a zero nella variabile relativa alla presenza di tecnologia *wireless*. Il centroide del gruppo 2, invece, mostra un valore prossimo ad uno anche in questa variabile. A questi due gruppi afferiscono i modelli con le migliori caratteristiche tecniche (ed i prezzi più elevati). Il gruppo 1 si differenzia dal 2 per l'assenza di tecnologia *wireless* nei modelli afferenti. I gruppi 5 e 6 riuniscono i modelli con i prezzi più bassi, un numero di casse ed una potenza inferiori alla media e, in genere, l'assenza di tecnologia *wireless*, dvd *recorder*, e dvx. Il gruppo 5 si differenzia dal 6 per la presenza di *radio data system* nei modelli afferenti. I modelli dei gruppi 3 e 4 mostrano caratteristiche «anomale» e intermedie fra le due situazioni estreme analizzate. Il modello nel gruppo 3, a fronte di valori inferiori alla media nelle altre variabili, si caratterizza per la presenza del dvd *recorder* (sempre assente nei modelli con basso prezzo e potenza e numero di casse inferiori alla media). Il gruppo 4 si caratterizza, invece, per un numero di casse superiore alla media, la presenza di dvd e dvx ma l'assenza di *radio data system* (in genere presente anche nei modelli con caratteristiche tecniche inferiori).

In base alla mappa così ottenuta, sia l'azienda produttrice, sia il possibile acquirente, potranno vedere, per ogni modello, il gruppo di modelli maggiormente simili (quelli afferenti a gruppi contigui nella mappa). Ai gruppi confinanti apparterranno i principali *competitors* mentre ai neuroni non direttamente collegati nella mappa afferiranno modelli con caratteristiche tecniche molto diverse e, quindi, appartenenti sicuramente ad un altro segmento di mercato.

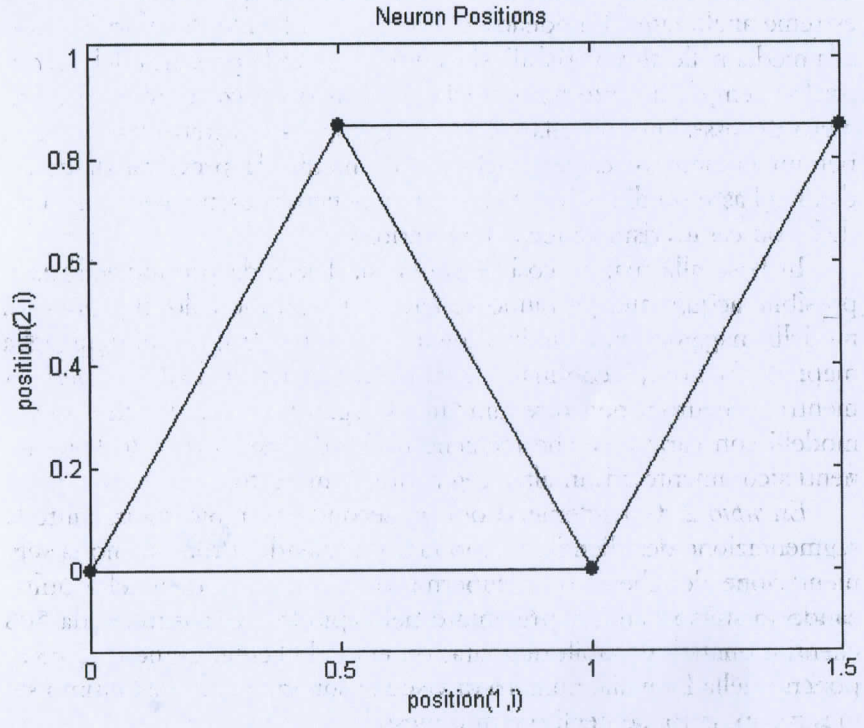
Esempio 2. Consideriamo ora un secondo esempio riguardante la segmentazione dei clienti che può fare un'azienda. Proponiamo la segmentazione dei clienti d'un supermercato con carta di fedeltà utilizzando lo stesso data set presentato nel capitolo IX (costituito da 508 clienti e quattro variabili quantitative: età del cliente, numero di componenti della famiglia, numero di visite al supermercato nell'ultimo semestre, spesa totale negli ultimi 6 mesi).

Come già sottolineato nel capitolo IX, le prime 2 variabili riguardano le caratteristiche del cliente e della sua famiglia (rilevate al momento del rilascio della *fidelity card*), mentre le altre due sono variabili

comportamentali. Nella tab. 9.14 sono riportate le statistiche descrittive delle 4 variabili e la matrice di correlazione tra esse, ottenute mediante SPSS con le procedure descritte nel primo e nel secondo capitolo.

Per effettuare la classificazione mediante le reti di Kohonen dobbiamo operare una serie di scelte. Quella fondamentale, come per la classificazione con le *k*-medie, riguarda il numero di gruppi, che deve essere fissato a priori. Si potrebbe cominciare con la partizione con 2 gruppi, valutarne la bontà, e passare eventualmente a quella con 3 gruppi, ripetendo la procedura dall'inizio, etc. A fini didattici, per poter confrontare la partizione finale con quella ottenuta con le *k*-medie e presentata nel capitolo IX, scegliamo 4 gruppi. Impostiamo quindi una rete di dimensione 2×2 con neuroni esagonali (l'architettura è riportata nella figura 12.17).

FIG. 12.17. *Architettura di rete utilizzata per la classificazione dei 508 clienti di un supermercato*



Con una tipologia esagonale, ogni neurone è connesso con tutti i neuroni adiacenti e non solo con quelli a destra, a sinistra, sopra e sotto.

In una mappa 2×2 ogni neurone sarà quindi connesso con tutti gli altri. Alcune scelte risultano «limitate» a causa della dimensione particolarmente ridotta della mappa: il raggio non può essere maggiore di uno e la funzione di intorno h_{rc} sarà una costante. Impostiamo una sola fase di addestramento di 1000 iterazioni, con un *learning rate* pari a 0.02 ed una distanza euclidea. Scegliamo i *reference vectors* iniziali estraendo casualmente dal data set 4 vettori riga.

Con meno di 500 iterazioni l'algoritmo raggiunge la convergenza (i *reference vector* rimangono immutati pur continuando l'addestramento). La soluzione finale (*reference vector* finali e osservazioni afferenti ad ogni neurone) rimane stabile cambiando i *reference vector* iniziali o il *learning rate* (passando da 0.02 a 0.03 o 0.04). Questo significa che molto probabilmente la rete è riuscita a raggiungere un «ottimo» assoluto e non locale. Nella tabella 12.3 sono riportati i *reference vector* ed i centrodi finali dei neuroni (si ricordi che le variabili sono espresse in termini di scostamenti standardizzati) ed il numero di casi (unità) in ogni neurone.

TAB. 12.3. *Reference vectors e centroidi (in termini di scostamenti standardizzati) ottenuti per la segmentazione dei 508 clienti di un supermercato*

neurone	*Reference vectors*				Centroidi				N. casi
	età cliente	componenti	visite	spesa totale	Età cliente	componenti	visite	spesa totale	
1	0.2892	0.3255	-0.1679	-0.0949	0.7666	0.8117	-0.4583	-0.3599	151
2	0.3159	0.3712	0.6436	0.6051	0.3583	0.5404	1.7113	1.6948	90
3	-0.2169	-0.2549	-0.4429	-0.4164	-0.5282	-0.5735	-0.6109	-0.5333	185
4	-0.3796	-0.4269	0.2206	0.1247	-0.6122	-0.7913	0.3327	-0.0005	82

- Nel primo gruppo (il neurone in basso a sinistra nella figura 12.17) vi sono 151 clienti, di età superiore alla media, con un numero di componenti della famiglia superiore della media, che effettuano un numero di visite ed una spesa totale di poco inferiori ai valori medi.
- Il secondo gruppo (il neurone in basso a destra nella figura 12.17) è formato da 90 clienti, leggermente più anziani della media, con famiglia di dimensioni abbastanza elevate e con valori delle variabili comportamentali superiori alla media. Questo gruppo è il più interessante per il supermercato poiché identifica i clienti con i valori più elevati del numero di visite e della spesa. Sia la loro età sia il numero di componenti sono maggiori della media. Tale segmento era stato individuato anche con l'utilizzo delle *k*-medie (si veda il capitolo IX).

• Il terzo gruppo, di 185 unità (il neurone in alto a sinistra nella figura 12.17) presenta valori inferiori alla media in tutte le quattro variabili. Si tratta dei clienti più giovani, con famiglia poco numerosa (probabilmente molti *single*), che spendono poco ed effettuano un numero di visite inferiore alla media. Anche questo segmento era stato individuato con la *cluster analysis* non gerarchica.

• Il quarto gruppo di 82 clienti (il neurone in alto a sinistra nella figura 12.17) è sempre formato da persone giovani e con un nucleo familiare di ridotte dimensioni. Nelle variabili comportamentali questi clienti si differenziano nettamente da quelli del gruppo precedente, poiché compiono una spesa totale nella media ed effettuano un numero di visite superiori alla media. Anche questo segmento risulta interessante per l'azienda: con azioni di marketing e promozioni mirate l'azienda dovrebbe riuscire a condurre le persone del gruppo 2 (simili per le caratteristiche demografiche) nel gruppo 3.

Confrontando la partizione ottenuta con quella individuata dalle *k*-medie, notiamo che sia le reti di Kohonen sia la cluster non gerarchica individuano due gruppi ben precisi di clienti: i giovani, che spendono meno e vengono poco al supermercato, ed il segmento dei clienti con famiglia più numerosa, di età leggermente superiore alla media, che vengono molto spesso e spendono molto più della media. Gli altri due gruppi identificano invece segmenti leggermente diversi nelle due analisi.

Una differenza evidente riguarda il valore dei centroidi. Quelli individuati dalle *k*-medie risultano ben distanti mentre i quattro centroidi individuati dalle reti di Kohonen risultano più simili. Questo è dovuto alla differente regola di apprendimento dei due modelli: nelle *k*-medie, ad ogni iterazione, il centroide viene aggiornato solo in base ai valori delle variabili dell'osservazione che afferisce al gruppo; nelle reti di Kohonen, invece, ad ogni iterazione, il *reference vector* viene aggiornato anche in base al valore dei *reference vectors* dei neuroni adiacenti.

Per verificare il grado di omogeneità dei gruppi ottenuti e quindi la validità della classificazione, possiamo utilizzare l'indice R^2. Il calcolo avviene in modo analogo a quanto riportato nel capitolo IX per la partizione ottenuta con le *k*-medie. Dalla tab. 12.4 si ottengono per le singole variabili i seguenti valori:

Punteggi: età cliente: $R^2 = 0.362$
Punteggi: n. componenti: $R^2 = 0.471$
Punteggi: n. visite: $R^2 = 0.737$
Punteggi: spesa: $R^2 = 0.653$

I 4 gruppi della partizione risultano quindi più omogenei nel loro interno con riferimento alle variabili comportamentali (numero di visite e spesa) e meno omogenei con riguardo all'età ed al numero di componenti della famiglia. Anche da questo punto di vista emergono delle differenze con la partizione ottenuta con le *k*-medie (in cui i gruppi erano più omogenei con riferimento al numero di componenti).

L'indice R^2 globale può essere calcolato come media semplice dei 4 indici delle singole variabili:

$$R^2 = (0.362 + 0.471 + 0.737 + 0.653)/4 = 0.555.$$

Il valore risulta leggermente inferiore a quello ottenuto con le *k*-medie, indicando una maggiore variabilità all'interno dei gruppi. La classificazione ottenuta con le *k*-medie sembra quindi leggermente «migliore». Non bisogna tuttavia dimenticare le informazioni aggiuntive fornite dalle reti di Kohonen date dalla dislocazione spaziale dei vettori nella mappa. In base alla architettura utilizzata, il gruppo 1 risulta più simile ai gruppi 2 e 3 e «diametralmente» opposto al gruppo 4. Quindi, i clienti appartenenti al gruppo 1 saranno maggiormente simili, come comportamento di acquisto e/o caratteristiche demografiche, a quelli dei gruppi 2 e 3. Saranno invece molto diversi da quelli appartenenti al gruppo 4.

La partizione con 4 gruppi è abbastanza soddisfacente, ma non si deve dimenticare che la variabilità all'interno dei gruppi (l'errore) è uguale al 44,5% della devianza totale. Per ottenere partizioni con gruppi più omogenei — e quindi classificazioni migliori — occorrerebbe aumentare il numero di gruppi e ripetere l'intera procedura dall'inizio.

Naturalmente, aumentando il numero di gruppi cresce anche il valore di R^2, poiché s'individuano *cluster* più omogenei. La scelta del numero «ottimo» di gruppi deve però tener conto, come nel caso della *cluster analysis*, dell'incremento di R^2 passando da g a $(g + 1)$ gruppi: se tale aumento è modesto, può essere preferibile la partizione meno fine con g gruppi, che offre una sintesi migliore.

Nell'esempio in esame, considerando una segmentazione dei clienti con 5 o 6 gruppi, i miglioramenti dell'indice di bontà globale R^2 non risultano molto sensibili, e quindi non giustificano da soli il passaggio a partizioni più disaggregate. Tuttavia, questo potrebbe risultare opportuno per meglio indirizzare le politiche di marketing a target specifici.

Le caratteristiche dei gruppi, descritte nella tab. 12.3 in termini di scostamenti standardizzati, possono venire meglio evidenziate considerando per ogni segmento le statistiche descrittive calcolate sui valori

originari. La procedura in SPSS è analoga a quella descritta nel capitolo IX per ottenere la tab. 9.9. Si considera l'analisi della matrice dei dati partizionata in base alla variabile *cluster di appartenenza*, che è stata aggiunta come colonna ulteriore. Nella tabella 12.4 sono riportate le statistiche descrittive per la segmentazione dei 508 clienti in 4 gruppi. Si lascia al lettore commentare tutte le informazioni che se ne possono trarre (sulla base delle specificazioni e dei commenti riportati per la tabella 12.3) e confrontare i risultati con quelli ottenuti nel capitolo IX.

TAB. 12.4. *Statistiche descrittive dei 4 segmenti dei 508 clienti di un supermercato*

Statistiche descrittive

Gruppo 1

	N	Minimo	Massimo	Media	Deviazione st.
età cliente	151	27	78	49.17	9.226
n. componenti	151	1	8	3.30	1.046
n. visite	151	1	10	3.84	2.488
spesa totale	151	12	642	208.14	135.313
Validi (listwise)	151				

Gruppo 2

	N	Minimo	Massimo	Media	Deviazione st.
età cliente	90	28	75	44.90	9.967
n. componenti	90	1	6	2.98	1.070
n. visite	90	8	41	20.67	7.713
spesa totale	90	445	3529	1424.87	785.044
Validi (listwise)	90				

Gruppo 3

	N	Minimo	Massimo	Media	Deviazione st.
età cliente	185	22	56	35.64	7.387
n. componenti	185	1	4	1.64	.734
n. visite	185	1	8	2.66	1.698
spesa totale	185	3	315	105.47	69.550
Validi (listwise)	185				

Gruppo 4

	N	Minimo	Massimo	Media	Deviazione st.
età cliente	82	22	53	34.76	6.805
n. componenti	82	1	3	1.37	.557
n. visite	82	4	26	9.98	3.979
spesa totale	82	120	1185	420.94	204.677
Validi (listwise)	82				

RIFERIMENTI BIBLIOGRAFICI

BELLACICCO, A. e LAURO, N.C. (a cura di) (1997), *Reti Neurali e Statistica*, Franco Angeli, Milano.

BISHOP, C.M. (1995), *Neural Networks for Pattern Recognition*, Clarendon Press, Oxford.

CHEN, S., COWAN, C.F.N. e GRANT, P.M. (1991), Orthogonal least squares learning algorithm for radial basis function networks, *IEEE Transactions on Neural networks*, vol. 2, n. 2, pp. 302-309.

EFRON, B. (1983), *The Jackknife, the Bootstrap and Other Resampling Plans*, Society for Industrial and Applied Mathematics, Philadelphia.

EFRON, B. e TIBSHIRANI, R.J. (1993), *An Introduction to the Bootstrap*, Chapman and Hall, New York.

FAHLMAN, S. (1989), Faster learning variations of back-propagation: an empirical study, *Proceedings of the 1988 Connectionist Models Summer School, San Mateo*, California, Morgan Kaufmann Publishers, pp. 38-51.

GENTLE, J.E. (1998), *Numerical Linear Algebra for Applications in Statistics*, Springer, New York.

HORNIK, K., STINCHCOMBE M. e WHITE, H. (1990). Universal approximation of an unknown mapping and its derivatives using multilayer-feed-forward networks, *Neural Networks*, vol. 3, pp. 551-560.

INGRASSIA, S. e DAVINO C. (a cura di) (2002), *Reti Neuronali e Metodi Statistici*, Franco Angeli, Milano.

INGRASSIA, S. e MORLINI I. (2002), Modelli neuronali per piccoli insiemi di dati, in: Lauro C.N. e Scepi G. (a cura di), *Analisi Multivariata per la Qualità Totale*, Franco Angeli, Milano, pp. 29-40.

INGRASSIA, S. e MORLINI I. (2005), Neural network modelling for small data sets, *Technometrics*, vol. 47, 3, pp. 297-312.

KOHONEN, T. (1997), *Self Organizing Maps*, Springer, Heilderberg.

MACKAY, D.J.C. (1992), Bayesian interpolation, *Neural Computation*, vol. 4, n. 3, pp. 415-447.

MOODY, J.E. (1992), *The Effective Number of Parameters: An Analysis of Generalization and Regularization in Non-Linear Learning System*, in: *Advances in Neural and Information Processing Systems*, vol. IV, Morgan and Kaufmann, San Mateo, CA.

MORLINI, I. (1998), Influenza dell'addestramento sulla rappresentazione dei dati nelle reti di Kohonen: un'applicazione, *Atti della XXXIX Riunione Scientifica della SIS*, Sorrento, pp. 125-126.

MORLINI, I. (1999), Radial basis function networks with partially classified data, *Ecological Modelling*, Elsevier Science, vol. 120, pp. 109-118.

MORLINI, I. (2001), Using radial basis function networks for classification problems, in: Borra S. *et al.* (a cura di), *Advances in Classification and Data Analysis*, Springer, Berlin, pp. 119-126.

MORLINI, I. (2002a), Le reti con funzione a base radiale, in: Ingrassia, S. e Davino, C. (a cura di), *Reti Neuronali e Metodi Statistici,* Franco Angeli, Milano, pp. 137-158.

MORLINI, I. (2002b), Apprendimento in due fasi per reti RBF, in: Ingrassia, S. e Davino, C. (a cura di), *Reti Neuronali e Metodi Statistici,* Franco Angeli, Milano, pp. 159-174.

MORLINI, I. (2004), Reti neurali e data mining per l'analisi della *customer satisfaction*: il caso della qualità della didattica nella Facoltà di Economia di Parma, in: Davino, C. e Lauro, C.N. (a cura di), *Analisi Simbolica e Data Mining*, Franco Angeli, Milano, pp. 173-203.

MORLINI, I. e ORLANDINI, S. (2001), Multivariate analysis of radar images for environmental monitoring,*Metron,* vol. LIX, n. 1-2, pp. 169-189.

ORR, M.J.L. (1993), *Regularized centre recruitment in radial basis function networks*, Research Paper, 59, Centre for Cognitive Science, Edinburgh University.

RIPLEY, B.D. (1996), *Pattern Recognition and Neural Networks*, Cambridge University Press, Cambridge.

SMITH, M. (1996), *Neural Networks for Statistical Modeling*, International Thompson Computer Press, USA.

SPIEGELHALTER, D.J. DAWID, A.P. LAURITZEN, S.L. e COWELL R.J. (1993), Bayesian analysis in expert system (with discussion), *Statistical Science vol. 8*, pp. 219-283.

STONE, M. (1974), Cross-validatory choice and assessment of statistical predictor, *Journal of the Royal Statistical Society B*, vol. 36, pp. 111-147.

THISTED, R.A. (1988), *Elements of Statistical Computing. Numerical Computation*, Chapman and Hall, New York.

WHITE, H. (1989), Some asymptotic results for learning in single hidden layer feedforward network models, *Journal of the American Statistical Association*, vol. 84, pp. 1003-1013.

WHITE, H. (1992), *Artificial Neural Networks. Approximation and Learning Algorithms*, Blackwell, Cambridge, Massachussets.

TAVOLE