UNIVERSITÀ DEGLI STUDI DI SALERNO

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI
CORSO DI LAUREA IN SCIENZE DELL'INFORMAZIONE

TESI DI LAUREA

CLASSIFICAZIONE STELLE/GALASSIE TRAMITE RETI NEURALI

RELATORI
PROF. R. TAGLIAFERRI
PROF. G. LONGO

CANDIDATO
NICOLA CAPUANO

ANNO ACCADEMICO 1996/97

INTRODUZIONE

Questo lavoro di Tesi si inserisce nell'ambito di CRoNaRio: una recente estensione di un progetto a lungo termine iniziato al Caltech nel 1993 e finalizzato alla costruzione del Palomar Norris Sky Catalog: un catalogo completo di tutti i corpi celesti presenti nella Second Palomar Sky Survey (POSS-II), che copre tutto l'emisfero Nord della sfera celeste, per un numero stimato di 2×10^9 *stelle e* 5×10^7 *galassie. Le lastre fotografiche che costituiranno la POSS-II vengono attualmente acquisite tramite il telescopio Schmidt del monte Palomar e, dopo essere state digitalizzate e linearizzate, vengono processate con il software SKICAT (sviluppato al Caltech) che provvede all'estrazione automatica dei cataloghi.*

SKICAT è attualmente il miglior software esistente per la generazione di cataloghi. Esso consta di una fase di segmentazione (nella quale identifica gli oggetti sulla lastra separandoli dal fondo altamente rumoroso) e di una fase di classificazione (nella quale, per ogni oggetto estratto, stabilisce la classe stella/galassia di appartenenza). Il presente lavoro di Tesi si propone di costruire un software che, tramite l'utilizzo di Reti Neurali, sia in grado di ottenere prestazioni migliori di SKICAT ovvero che riesca a:

- *identificare gli oggetti con maggiore precisione riducendo al minimo la generazione di oggetti spuri (oggetti creati artificialmente a causa di anomalie di luminosità della lastra);*
- *classificare gli oggetti in maniera più accurata.*

L'accuratezza della separazione stelle/galassie determina infatti il vero limite della utilizzabilità scientifica dei dati ricavati dalle survey. Per molti programmi di processing dei dati è necessario ottenere, in questa fase, un'accuratezza maggiore del 90%. Ogni miglioramento, anche piccolo, nella percentuale di classificazioni corrette determinerebbe un salto di qualità nell'utilizzo scientifico dei cataloghi.

I capitoli che seguono sono idealmente divisi in tre sezioni:

- *nella prima sezione (capitolo 1) si forniranno dettagli sul progetto CRoNaRio e sul software SKICAT col quale ci si vuole confrontare, si descriverà inoltre, per sommi capi, l'approccio alternativo proposto;*

- *nella seconda sezione (capitoli 2, 3, 4, 5) verranno date le basi concettuali necessarie alla comprensione dei capitoli successivi e si descriveranno tutti i modelli neurali utilizzati nel lavoro (alcuni di questi sono originali);*
- *nella terza ed ultima sezione (capitoli 6, 7, 8) varrà analizzata, passo per passo, la procedura proposta presentando i relativi esperimenti e confrontando ogni risultato intermedio con SKICAT.*

INDICE

CAPITOLO 8
Terzo Passo: Classificazione

CAPITOLO 1

Il progetto CRoNaRio.

1.1 Introduzione.

Nell'ultimo decennio si è assistito ad una vera e propria rivoluzione nelle tecniche di acquisizione dei dati astronomici. Lo sviluppo di nuove emulsioni fotografiche (con maggiore sensibilità e risoluzione) e di rivelatori digitali a largo campo (CCD o *Charge Coupled Devices*) hanno enormemente accresciuto la quantità e la qualità dei dati a disposizione della comunità astronomica internazionale. Telescopi equipaggiati con camere CCD a largo campo possono produrre fino a 2 Gb di dati per notte. Dati che, molto spesso, devono essere resi fruibili dalla comunità scientifica in tempo quasi reale, cioè al più entro uno o due giorni dalle osservazioni.

La *pre-riduzione*, la *calibrazione* e l'*estrazione* delle informazioni dai dati grezzi richiedono, di contro, procedure complesse che, se effettuate con tecniche tradizionali (interattive) possono risultare estremamente *time consuming*. Caso estremo di questo problema sono le cosiddette ***sky survey***: atlanti fotografici o digitali di ampie porzioni del cielo o di interi emisferi celesti quali, ad esempio, la Prima e la Seconda *Palomar Sky Survey* (POSS-I e POSS-II), la *Sloan Digital Sky Survey* (SLOAN-DSS) o la *ESO Imaging Survey* (EIS).

Il problema principale posto dalle *survey* consiste nell'estrazione, a partire dalle immagini ottenute dai telescopi, di *cataloghi* di corpi celesti (vedi figura 1.1), ovvero di tabelle in cui ogni riga rappresenta un oggetto e ogni colonna un particolare parametro (frutto di misurazioni effettuate nello spazio dei pixel) dell'oggetto stesso. Quest'operazione permette di traslare l'informazione dallo spazio dei pixel (rumoroso e altamente correlato) ad uno spazio con un minor numero di dimensioni dove la quantità di rumore è drasticamente ridotta. Uno dei passi più importanti in quest'operazione di trasposizione da pixel a cataloghi è la discriminazione tra oggetti non estesi (cioè oggetti il cui profilo è identico alla *Point Spread Function* strumentale) ed oggetti estesi. Per convenzione, in letteratura, quando ci si riferisce a questa operazione si parla di *separazione stelle/galassie*.

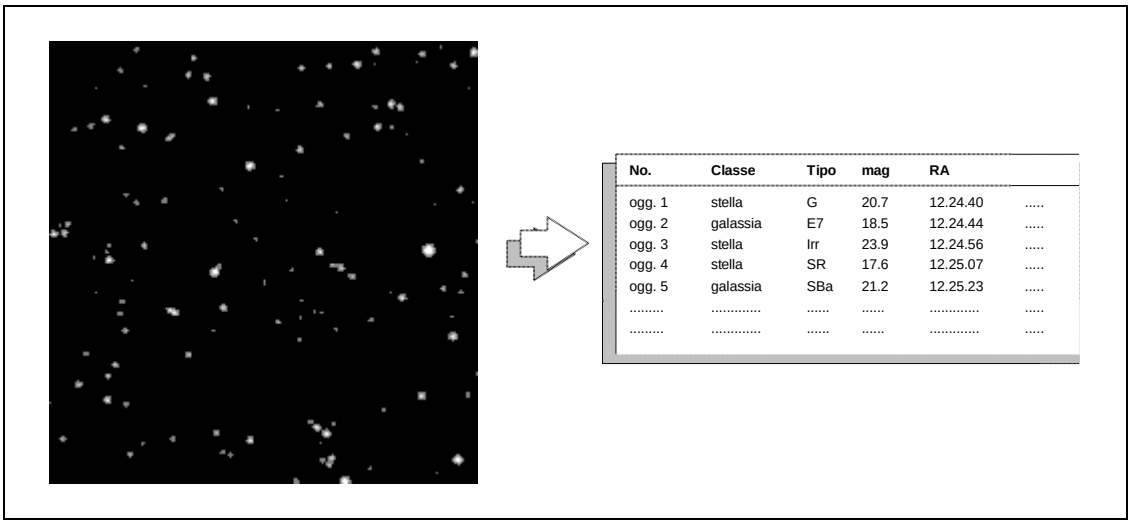

Fig. 1.1 - esempio di generazione di un catalogo da un campo stellare.

L'accuratezza della separazione stelle/galassie determina, specialmente a bassi livelli di flusso, il vero limite dell'utilizzabilità scientifica dei dati ricavati dalle *survey*. Per molti programmi di processing dei dati (per esempio per la ricerca di *quasars* di alto *redshift*) è necessario ottenere, in questa fase, un'accuratezza maggiore del 90%. Ogni miglioramento, anche piccolissimo, nella percentuale di classificazioni corrette determina un salto di qualità nell'utilizzo scientifico dei cataloghi.

L'estrema sensibiltà raggiunta sia dalle lastre fotografiche che dai CCD, permette di ottenere immagini nelle quali risultano evidenti anche oggetti a bassissima luminosità (si veda la differenza di risoluzione tra le due immagini di figura 1.2). Finora, tuttavia, i metodi computazionali esistenti per la classificazione delle immagini dovevano operare una scelta tra rivelare gli oggetti brillanti (scarsa completezza ed alta attendibilità) o rivelare gli oggetti deboli (alta completezza e bassa attendibilità). I recenti sviluppi sia nel campo del *pattern recognition* che del *machine learning*, rendono oggi possibile la messa a punto di procedure atte a costruire in modo automatico cataloghi che classificano correttamente sia oggetti deboli che oggetti brillanti.

Una di queste procedure è lo SKICAT (*SKy Image Cataloging and Analysis Tool*) sviluppato da un gruppo di ricerca del *Caltech* di Pasadena nell'ambito del progetto CRoNaRio. Di tale progetto si parlerà dettagliatamente nel paragrafo 1.2 mentre il paragrafo 1.3 sarà dedicato ad una descrizione sommaria dello SKICAT.

Sebbene SKICAT possa considerarsi rivoluzionario sotto diversi aspetti, esso è passibile di sensibili miglioramenti. Il presente lavoro di tesi si propone di utilizzare un approccio totalmente nuovo per la risoluzione del problema del riconoscimento e della classificazione di oggetti celesti basato sull'utilizzo di reti neurali. Tale approccio sarà accennato nel paragrafo 1.4 ed approfondito nei successivi capitoli. Dimostreremo nei capitoli 7 e 8 che i risultati ottenuti con questo nuovo metodo sono superiori sia a quelli di SKICAT che a quelli di altri prodotti software affini.

1.2 Finalità del progetto CRoNaRio.

CRoNaRio (Caltech - Roma - Napoli - Rio de Janeiro) è una recente estensione di un progetto a lungo termine iniziato al Caltech (*California Institute of Technology*) già nel 1993 e finalizzato alla costruzione del *Palomar Norris Sky Catalog* (PNSC): un catalogo completo di tutti gli oggetti presenti nella *Second Palomar Sky Survey* (POSS-II) per un numero stimato di 2×10^9 stelle e 5×10^7 galassie. La POSS-II è realizzata in tre bande fotometriche e copre tutto l'Emisfero Nord (Equatore celeste incluso) della sfera celeste. Per ogni banda si hanno 894 lastre fotografiche di 6.5°×6.5° i cui centri sono distanziati l'uno dall'altro di 5° (si ha dunque una sovrapposizione di 0.75° per lato).

Le lastre vengono attualmente acquisite con il telescopio Schmidt del *Palomar Observatory* e digitalizzate da due gruppi di ricerca dello *US Naval Observatory* e dello *Space Telescope Science Institute* (STScI) in matrici di 23040×23040 pixel a 2 byte/pixel per un totale di circa 1 Gb di dati per lastra. Lo STScI provvede anche a derivare la *soluzione astrometrica di lastra*, cioè le equazioni di trasformazione tra le posizioni relative dei singoli pixel ed il sistema di coordinate assoluto (*ascenzione retta* e *declinazione*). Il completamento delle prime due fasi (acquisizione e digitalizzazione delle lastre) porterà alla costituzione di un archivio di circa 3 Tb di pixel di dati: il DPOSS (lo stato attuale della digitalizzazione è mostrato in figura 1.3).

Ogni lastra digitalizzata, prima di essere utilizzata per fini scientifici, deve essere calibrata in intensità (si devono cioè convertire le *densità* fotografiche in *intensità* assolute). Ciò viene effettuato in due fasi. Innanzitutto si provvede alla linearizzazione (calibrazione fotometrica relativa) dei dati tramite l'utilizzo di *spot* di calibrazione all'uopo registrati su ogni lastra e, quindi, alla calibrazione fotometrica assoluta tramite il confronto con frame CCD di campi stellari appositamente acquisiti.

Per l'acquisizione dei frame CCD di calibrazione sono attualmente utilizzati numerosi telescopi: il telescopio Oshkin (1.5 m) del Monte Palomar, i telescopi Danish (1.54 m) e Dutch (0.9 m) dell'ESO (*European Southern Observatory*) ed il telescopio

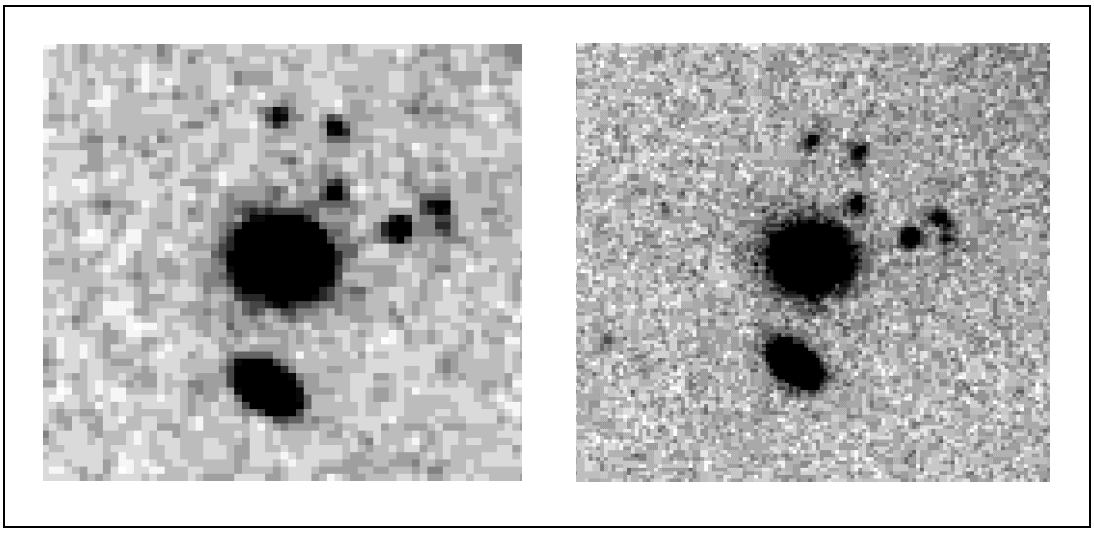

Fig. 1.2 - immagini POSS-I (sinistra) e POSS-II (destra) delle stesse galassie.

TT1 (1.5 m) dell'OAC (*Osservatorio Astronomico di Capodimonte*).

Il *processing* (cioè l'estrazione dei cataloghi) ha luogo sulle lastre digitalizzate e linearizzate (ma non calibrate in modo assoluto) tramite SKICAT presso quattro centri di ricerca: il Caltech, l'OAC, l'OAR (*Osservatorio Astronomico di Roma*) e l'ONRJ (*Observatorio Nacional in Rio de Janeiro*) con macchine e software identico (le macchine sono *workstation Ultra Sun* con 128/256 Mb di RAM ed un minimo di 2/6 Gb di memoria di massa, dotate di sistema operativo *Solaris*).

La riduzione delle lastre è un lavoro insidioso che impiega una grande quantità di tempo (la figura 1.4 mostra lo stato attuale di questo processo) ma il vero collo di bottiglia del progetto è l'allocazione delle ore di telescopio per l'acquisizione dei frame CCD di calibrazione. Con l'attuale andamento è ragionevole aspettarsi che l'insieme totale delle lastre non verrà calibrato prima della fine del 2001.

La gestione dei cataloghi viene effettuata con il software commerciale SYBASE che ne permette la completa manipolazione, compreso il confronto e il *merging* con altri cataloghi della stessa *survey*. I vari cataloghi vengono via via uniti tra loro a formare il *matched catalogue* che contiene tuttavia solo un sottoinsieme dei parametri presenti nei cataloghi iniziali (circa 50) e conserva memoria del modo in cui ogni dato è stato acquisito o di come esso è stato derivato. Il sistema è disegnato in modo da gestire un *database* continuamente ampliato ed aggiornato. Una volta completato, il *matched catalogue* costituirà il PSNC.

L'utilità del PSNC sarà grandissima per ricerche di: struttura della Galassia, struttura su larga scala dell'universo, identificazione automatica di controparti ottiche di sorgenti rilevate ad alte lunghezze d'onda, derivazione di cataloghi automatici di gruppi e ammassi di galassie, identificazione di oggetti peculiari (QSO, BL, Lac, ecc.), ricerca di stelle variabili, novae e supernovae ecc. Esso sarà anche utilizzato (il nostro caso)

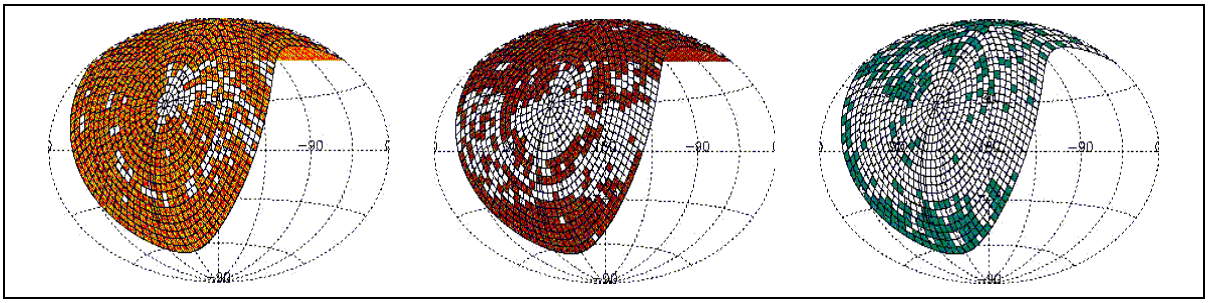

Fig. 1.3 stato attuale del processo di digitalizzazione delle lastre per le bande b, r, vi.

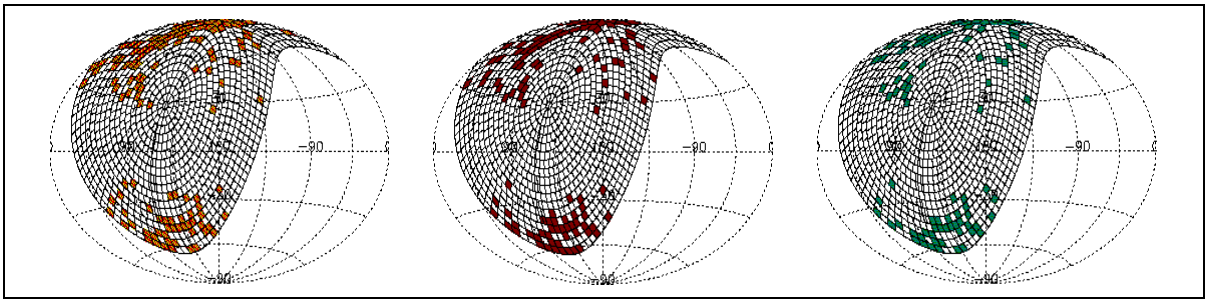

Fig. 1.4 stato attuale del processo di classificazione delle lastre per le bande b, r, vi.

per sperimentare nuovi algoritmi per l'elaborazione e per l'estrazione delle immagini.

Resta ancora da dire che, oltre ai pacchetti per la costruzione e la gestione del catalogo, il software di CRoNaRio comprenderà anche un pacchetto di moderni strumenti per l'analisi dei dati con sezioni per l'analisi multivariata, strumenti Bayesiani, reti neurali ecc. Questo pacchetto però, al momento in cui scriviamo, è ancora in fase di costruzione.

1.3 Una panoramica di SKICAT.

Lo *SKy Image Cataloging and Analysis Tool* è stato sviluppato al Caltech per facilitare la creazione e l'utilizzo di cataloghi presi da immagini larghe e sovrapposte, con particolare attenzione per le immagini POSS-II. Gli obiettivi delle *utility* software di cui SKICAT si compone cadono in tre grandi categorie: costruzione, gestione e analisi dei cataloghi. In questo paragrafo focalizzeremo la nostra attenzione sugli aspetti tecnici del primo di questi tre obiettivi che sarà anche argomento di questo lavoro di tesi: la ***costruzione automatica dei cataloghi***.

Per effettuare il processing di una lastra, l'utente utilizza la procedura automatizzata *AutoPlate* di SKICAT che legge da memoria di massa (tape da 8-mm) l'immagine e, dopo averla scomposta in una matrice di 13×13 *footprint* parzialmente sovrapposti, ciascuno di 2048^2 pixel, restituisce in output (dopo diverse ore di lavoro) il catalogo estratto dalla lastra (un file SYBASE). Vediamo ora, senza addentrarci nei dettagli,

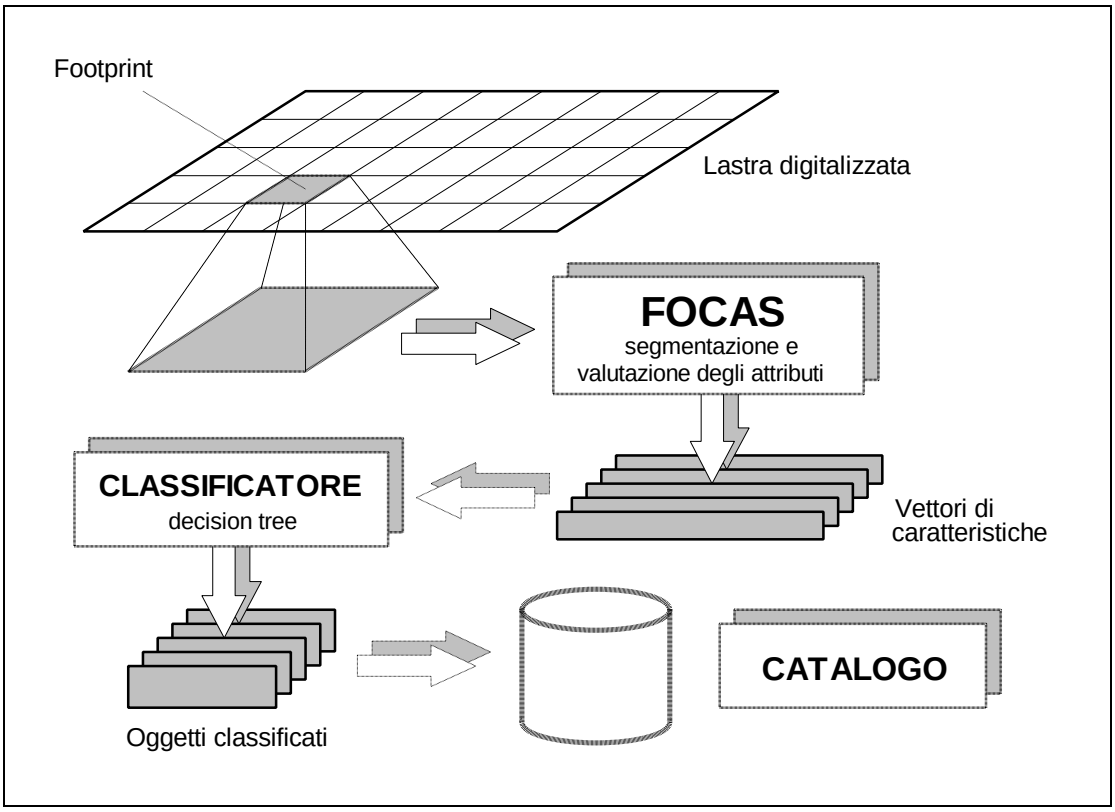

Fig. 1.5 - architettura del programma AutoPlate di SKICAT.

come agisce *AutoPlate* (la figura 1.5 ne mostra schematicamente l'architettura).

Il primo passo consiste nell'identificazione degli oggetti presenti sull'immagine (fase di ***segmentazione***) ed utilizza una versione opportunamente corretta del programma FOCAS (*Faint Object Classification and Analysis System*) di Jarvis e Tyson (vedi riferimenti del capitolo). Questa procedura fa scorrere sull'immagine una finestra di 5×5 pixel e confronta, ad ogni iterazione, la densità media calcolata nella finestra con la *densità locale* della lastra (densità dello sfondo valutata in quel punto). Se la densità media è più alta della *densità locale* di una soglia prestabilita, il pixel è considerato facente parte di un oggetto (viene attivato), altrimenti la densità media è utilizzata per aggiornare il valore della *densità locale* in quel punto. Dopo aver valutato tutti i punti dell'immagine, si procede a raggruppare in opportune strutture i punti attivati adiacenti. Ognuna di queste strutture raccoglierà quindi tutti i punti di un singolo oggetto.

Il secondo passo consiste nella ***valutazione degli oggetti*** rilevati al passo precedente. Si calcolano cioè, per ciascun oggetto, una serie di parametri fotometrici che ne permetteranno la corretta classificazione. I parametri calcolati dal FOCAS sono 7: il secondo e il quarto momento totale, l'ellitticità, l'intensità di picco, il raggio efficace, la convoluzione con un'immagine ideale di una stella e la magnitudine apparente. A questi, *AutoPlate* ne aggiunge una quarantina tra cui la magnitudine isofotale, centrale, di apertura e asintotica; l'area isofotale e totale; l'ellitticità; l'angolo di orientamento; i momenti pesati e non pesati; le posizioni di picco, intensità pesata e non pesata; ecc.

Il terzo e ultimo passo consiste nella ***classificazione degli oggetti*** in base ai parametri misurati. Si utilizza a tale scopo un software di induzione di tipo *decision-tree* (il GID3* di Fayyad e Irani), insieme ad un altro programma (il *Ruler* di Fayyad) per trasformare sistemi multipli di *decision-tree* in un solo insieme di regole di classificazione. Questi algoritmi lavorano utilizzando un *training set* costituito da oggetti correttamente classificati per dedurre un insieme di regole efficienti di tipo *if...then* atte alla classificazione di quegli stessi esempi. Sono stati creati insiemi di regole separati per ogni banda (*blu, rosso, vicino all'infrarosso*) a partire da immagini ottenute da CCD (*Charge-Coupled Device*). Questo supporto viene preferito, in questa fase, perché assicura una qualità d'immagine superiore ed un più alto rapporto segnale/rumore a qualsiasi livello di magnitudine rispetto alle normali lastre fotografiche. Dopo la fase di training (effettuata una volta sola al Caltech), gli algoritmi hanno generato un insieme di regole che è stato incorporato in *AutoPlate* ed è utilizzato per la classificazione degli oggetti di ogni lastra presentata. Il livello di accuratezza nella separazione stelle/galassie raggiunto con questo sistema è di circa il 90% fino a magnitudine 21 mag ovvero circa 1 mag in più rispetto a quanto si riusciva a fare in passato. Ciò comporta un raddoppio del numero di galassie disponibili per analisi scientifiche rispetto ai precedenti tentativi di *survey* automatizzati.

1.4 Un approccio alternativo.

Descriveremo ora il metodo adottato in questo lavoro di tesi per risolvere lo stesso problema di *AutoPlate* ovvero la costruzione automatica di un catalogo a partire da campi stellari digitalizzati. I concetti qui esposti in maniera non esaustiva, verranno adeguatamente approfonditi nei successivi capitoli così, allo scopo di facilitare l'approfondimento di punti specifici, è stato aggiunto ad ogni argomento il riferimento al capitolo ad esso dedicato. Come *AutoPlate*, anche il nostro programma, la cui architettura è sintetizzata in figura 1.6, è suddiviso in tre fasi: *segmentazione*, *valutazione degli oggetti* e *classificazione*.

La fase di ***segmentazione*** (vedi capitolo 6) consiste, al solito, nell'identificazione degli oggetti dell'immagine. Abbiamo utilizzato allo scopo delle *reti neurali non supervisionate* (vedi capitolo 4) che apprendono autonomamente come distinguere i pixel di fondo dai pixel appartenenti agli oggetti. L'idea è di scandire l'intera immagine evitando però di conservare come unica informazione il valore del singolo pixel. Per ogni pixel si considerano infatti tutti i valori dei punti appartenenti ad una finestra quadrata $n \times n$ centrata su di esso. Questo perché la classe a cui appartiene un punto (sfondo/oggetto) dipende in grande misura dal contesto in cui esso si trova (un pixel molto luminoso in un contesto di pixel scuri è quasi certamente rumore che non fa parte di alcun oggetto). I valori della finestra $n \times n$ costituiscono i *pattern* sui quali viene fatta operare la *rete neurale* (vedi capitolo 2).

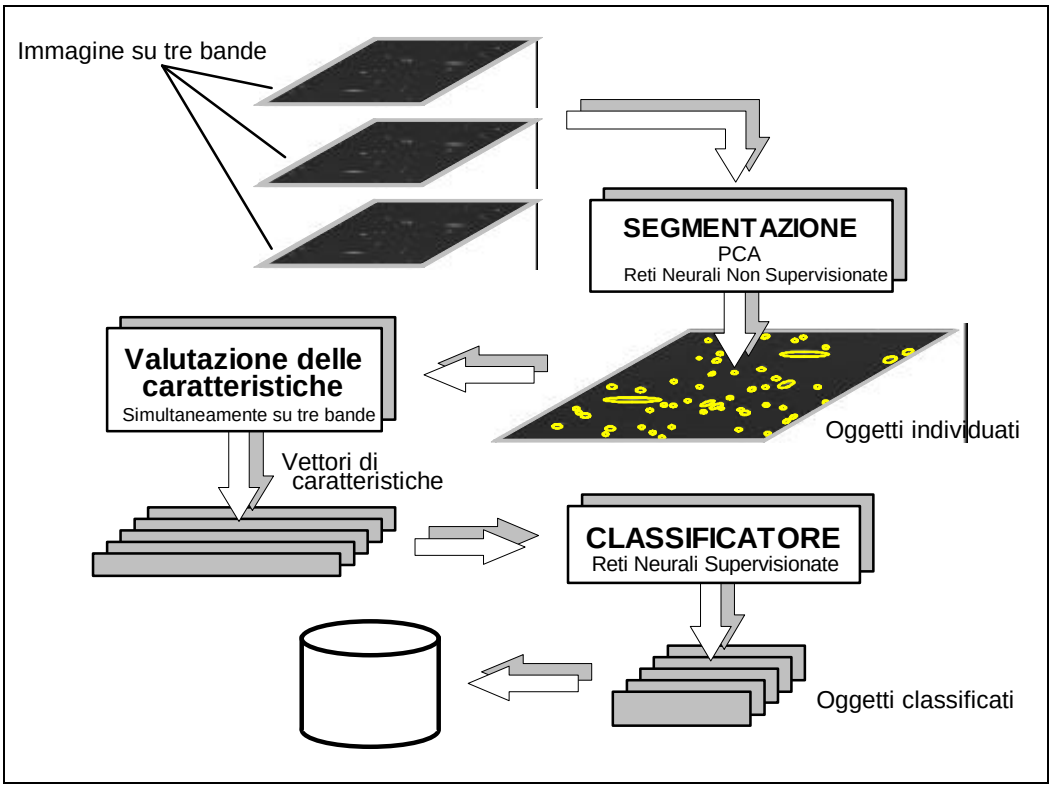

Fig. 1.6 - architettura del nostro programma.

È nostra intenzione sperimentare una segmentazione che lavori simultaneamente sulle tre bande dell'immagine. Ogni punto verrebbe in questo caso ad essere rappresentato dai valori di tre finestre, una per banda. Ciò porta ad avere *pattern* con troppe dimensioni per poter essere agevolmente gestite da una qualsiasi *rete neurale* (con finestre 7×7 si avrebbero *pattern* di 147 componenti). Abbiamo ovviato a questo inconveniente utilizzando delle tecniche di *analisi delle componenti principali* (vedi capitolo 5) tramite le quali si riesce a ridurre drasticamente la dimensionalità dei *pattern*.

Alla fase di *segmentazione* (che, in sostanza, colora ogni pixel in bianco o nero a seconda se questo faccia parte o meno di un oggetto) segue quella di ***valutazione degli oggetti*** (vedi capitolo 7) nella quale si individuano sulla lastra gli oggetti come blocchi di pixel attivati adiacenti e si calcolano i parametri fotometrici che andremo ad utilizzare nella fase successiva. I parametri sono stati scelti tra quelli maggiormente utilizzati in letteratura e su questi sono state utilizzate tecniche di *feature selection* per identificare quelli che meglio riescono a separare stelle e galassie.

In fase di segmentazione capita molto spesso che oggetti distinti vengano a trovarsi tanto vicini da non mostrare una vera e propria separazione tra i punti dell'uno e dell'altro. In questo caso sia *AutoPlate* che il nostro programma rilevano il blocco come se fosse un solo oggetto e poi procedono (effettuando opportune considerazioni sui parametri rilevati) alla separazione. Il difetto di *AutoPlate* (ereditato da FOCAS) è che la routine di separazione si attiva anche per oggetti singoli se questi sono molto grandi, creando una marea di *oggetti spuri* (ai quali non corrispondono oggetti reali).

La stessa cosa viene evitata agilmente nel nostro programma con risultati tali da surclassare *AutoPlate* su lastre con oggetti a bassa magnitudine (che sono poi le più utili ai fini della ricerca). Per altri tipi di oggetti, i risultati delle due procedure sono pressoché equivalenti.

Si procede infine alla ***classificazione degli oggetti*** (vedi capitolo 8) in base ai parametri rilevati in fase di valutazione. Essa viene effettuata tramite l'utilizzo di *reti neurali supervisionate* (vedi capitolo 3) fatte precedentemente apprendere su parti di altri cataloghi opportunamente scelti in base alla loro precisione (si utilizzano solo le classificazioni certe) in modo da coprire tutte le tipologie di oggetti celesti. I risultati della classificazione sono ottimi anche grazie alla più accurata separazione degli oggetti rispetto a quanto non faccia SKICAT.

Come sarà ormai chiaro, scopo finale di questo lavoro di tesi è stato la produzione di un ***software*** in grado di effettuare la generazione automatica di cataloghi e che possa sostituire l'*AutoPlate* di SKICAT nell'ambito del progetto CRoNaRio. Il *software* è stato scritto per massima parte in linguaggio C++ standard ed è, quindi, ampiamente portabile. È stato necessario aggiungere alcune *routine* in SPP (un linguaggio *c-like*) per poter interfacciare il programma con gli altri strumenti di analisi utilizzati nell'ambito di CRoNaRio. Il tutto è stato compilato su macchine *Ultra Sun* e gira sotto sistema operativo *Solaris*.

Riferimenti.

S. Andreon, S. Zaggia, R. de Carvalho, S. Djorgovski *et al.* "The CRONA (Caltech-Roma-Napoli) project" in *astro-ph/*9706238, accettato in giugno 1997.

U. M. Fayyad, S.G. Djorgovski, N. Weir "From Digitized Images to On-line Catalogs: A Machine Learning Application" in *AI Magazine*, accettato in aprile 1996.

J. F. Jarvis, J. A. Tyson "FOCAS: Faint Object Classification And Analysis System" in *The Astronomical Journal*, vol. 86, no. 3, pp. 476-495, 1981.

N. Weir, U. M. Fayyad, S. G. Djorgovski, J. Roden "The SKICAT System for Processing and Analyzing Digital Imaging Sky Surveys" in *Publications of the Astronomical Society of the Pacific*, no. 107, pp. 1243-1254, 1995.

N. Weir, U. M. Fayyad, S. G. Djorgovski "Automated star/galaxy classification for digitized POSS-II" in *The Astronomical Journal*, vol. 109, no. 6, pp. 2401-2414, 1995.

CAPITOLO 2

Reti Neurali e Pattern Recognition.

2.1 Introduzione.

Nel precedente capitolo abbiamo nominato alcune tecniche utili per la risoluzione in automatico del problema del riconoscimento di entità cosmiche quali le stelle e le galassie. Abbiamo parlato di *decision-tree* (usati da SKICAT) e di *reti neurali* (il nostro approccio). L'idea di base per la risoluzione di questo problema è quella di riprodurre su computer, in qualche modo, le conoscenze di un astronomo esperto. Un primo sistema potrebbe essere quello di creare un *database* di regole *if...then* ma il lavoro di traslazione risulterebbe eccessivo, costoso e poco affidabile se fatto manualmente. Sarebbe più opportuno creare un sistema che apprenda queste regole autonomamente a partire da esempi di classificazioni corrette o sbagliate.

L'***apprendimento da esempi*** riveste un ruolo fondamentale nel processo di *comprensione* dell'uomo (nei neonati ad esempio l'apprendimento avviene per imitazione, prove ed errori): il discente impara sulla base di casi specifici, non di teorie generali. In sostanza, l'apprendimento da esempi è un processo di ragionamento che porta all'individuazione di regole generali a partire da osservazioni di casi specifici: si parlerà, in questo caso, di *inferenza induttiva* in contrapposizione con l'*inferenza deduttiva* che consiste nel passare da conoscenze generali a casi particolari.

Due sono le caratteristiche tipiche del processo di apprendimento da esempi: in primo luogo la conoscenza appresa è *più compatta* dell'equivalente forma con gli esempi espliciti, richiede quindi meno capacità di memoria; in secondo luogo, la conoscenza appresa contiene *più informazione* degli esempi osservati (essendo una generalizzazione è applicabile anche a casi mai osservati). L'*inferenza induttiva* ha anche caratteristiche poco simpatiche: mentre la *deduzione* è un processo che preserva la verità (partendo da una conoscenza vera, la *deduzione* ci porta a conoscenze vere), nell'induzione, partendo da un insieme di fatti veri o falsi, si arriva alla formulazione di una regola generale che non è sicuramente corretta. Questo tipo di processo non preserva quindi la verità ma la falsità, nel senso che basta una sola asserzione falsa per escludere sicuramente una regola. Un sistema induttivo offre quindi la possibilità di

generare in modo automatico delle conoscenze che però possono essere fallaci. La frequenza degli errori dipende fortemente da come è stato scelto l'insieme di esempi su cui far apprendere il sistema e quanto questo è rappresentativo dell'universo dei casi possibili.

Le *reti neurali* sono, tra gli strumenti capaci di apprendimento da esempi, quelle con maggiore capacità di generalizzazione, esse riescono a gestire facilmente anche situazioni non previste in fase di apprendimento. Si tratta di modelli computazionali che si ispirano direttamente al funzionamento del cervello e che, almeno per il momento, si trovano ancora allo stadio primitivo della ricerca (i primi studi sull'argomento iniziarono negli anni '40 ma fu solo negli anni '80 che cominciarono a nascere le prime applicazioni). Sono strumenti di questo tipo quelli che utilizzeremo nel presente lavoro di tesi e che, quindi, procederemo ad illustrare in questo e nei successivi tre capitoli.

2.2 Il neurone biologico.

Tutto il sistema nervoso animale è composto da un numero elevatissimo di unità fondamentali molto ben caratterizzate nella loro struttura e simili tra loro dette *neuroni* (si calcola che il cervello umano ne contenga circa dieci miliardi). Ognuno di questi neuroni è connesso con migliaia di altri e con questi comunica con segnali sia di tipo chimico che elettrico.

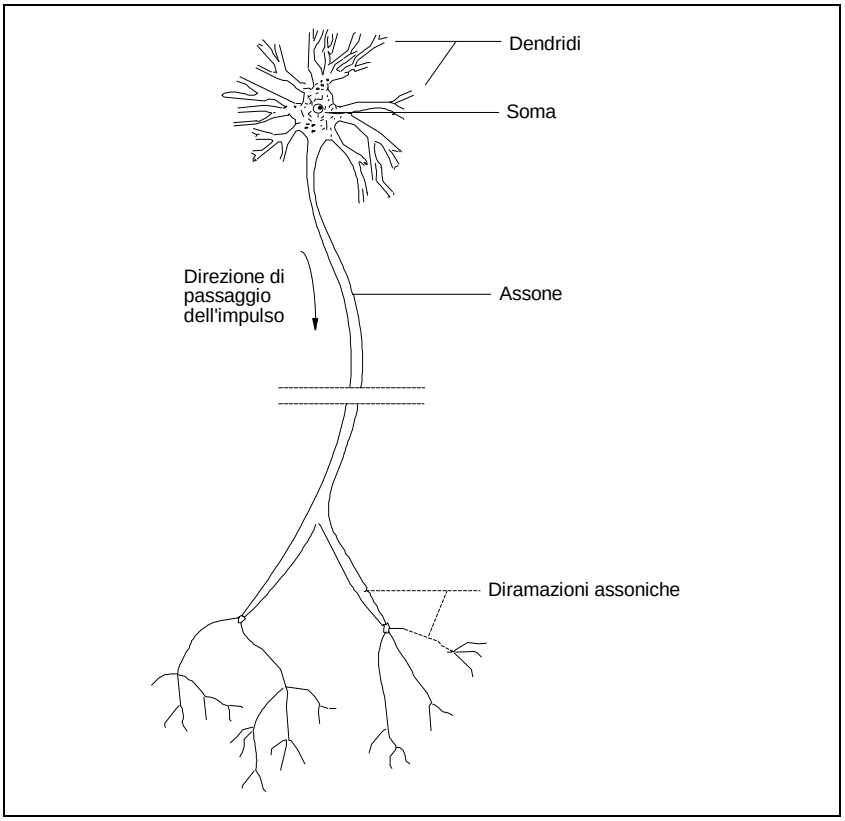

Fig. 2.1 - struttura di un neurone biologico.

Un neurone (come si può vedere in figura 2.1) è costituito da un corpo centrale detto *soma* da cui fuoriesce un prolungamento che prende il nome di *assone*. La fitta rete di interconnessione tra neuroni è realizzata dai *dendridi*, che sono delle ramificazioni che partono dal soma e vanno a raggiungere i terminali d'assone dei neuroni da collegare. Il punto dove un neurone è connesso con i dendridi provenienti da altri neuroni è detto *sinapsi*. Le sinapsi grazie all'azione di alcune sostanze chimiche dette *neurotrasmettitori*, possono assumere sia un'azione eccitatoria che inibitoria, per favorire o ostacolare il collegamento tra i neuroni a cui essa afferisce.

Ogni neurone riceve in ingresso segnali elettrici da tutti i dendridi e emette un impulso elettrico in uscita sull'assone. In pratica ogni neurone esegue una somma pesata dei segnali elettrici presenti sui suoi ingressi (che possono essere eccitatori o inibitori in funzione del collegamento sinaptico), se la *soglia caratteristica* è superata viene emesso un impulso in uscita. Le connessioni tra neuroni non sono rigide ma possono variare nel tempo, sia nell'intensità che nella topologia. Il peso del collegamento sinaptico, infatti, non è fisso ma deve essere variabile in quanto le sue variazioni sono alla base dell'apprendimento. Anche i collegamenti tra neuroni possono essere rimossi o se ne possono creare degli altri, questi infatti costituiscono la memoria della rete neurale.

2.3 Il neurone formale.

A seguito di ricerche effettuate negli anni '40 da W. McCulloch e W. Pitts vennero elaborati dei modelli matematici che riproducevano in maniera molto semplificata il funzionamento di un neurone ed è a questi modelli che si riconduce la teoria delle *reti neurali*. Il neurone formale (vedi figura 2.2) rispecchia nella struttura il funzionamento del neurone biologico. Agli ingressi binari $x_1 \dots x_n$ vengono infatti associati dei pesi $w_1 \dots w_n$ che possono essere positivi o negativi e ne viene effettuata la somma (pesata). Se il valore della somma supera un determinato livello di soglia θ l'uscita vale 1 altrimenti vale 0.

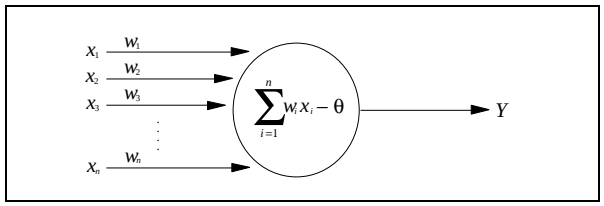

Fig. 2.2 - neurone formale.

In sé il modello del neurone non avrebbe molto senso se non fosse stato dimostrato che qualsiasi calcolo esprimibile tramite un programma per computer è realizzabile da un'appropriata rete di neuroni formali introducendo dei neuroni di input che prelevano i dati e neuroni di output che comunicano i risultati. Data inoltre la scarsa efficienza nei calcoli matematici, le reti neurali non avrebbero avuto il successo che hanno se non avessero proprietà tali da surclassare i calcolatori convenzionali in determinate

applicazioni. L'idea innovativa fu quella di rendere i pesi dei collegamenti modificabili con meccanismi di tipo biologico, in tal modo le reti neurali sono in grado di adattarsi a condizioni di funzionamento non previste in fase di progetto, cosa quasi indispensabile in numerosi casi (ad esempio nei sistemi di controllo o nel pattern recognition).

Le ***reti neurali*** sono, come dovrebbe essere ormai chiaro, costituite da uno strato di neuroni di *input*, uno strato di neuroni di *output* ed eventualmente uno o più strati intermedi detti *hidden* (vedi figura 2.3). Le interconnessioni vanno da uno strato al successivo e i valori dei segnali possono essere sia discreti che continui. I valori dei pesi associati agli input di ogni nodo possono essere statici o dinamici in modo tale da adattare plasticamente il comportamento della rete in base alle variazioni dei segnali d'ingresso.

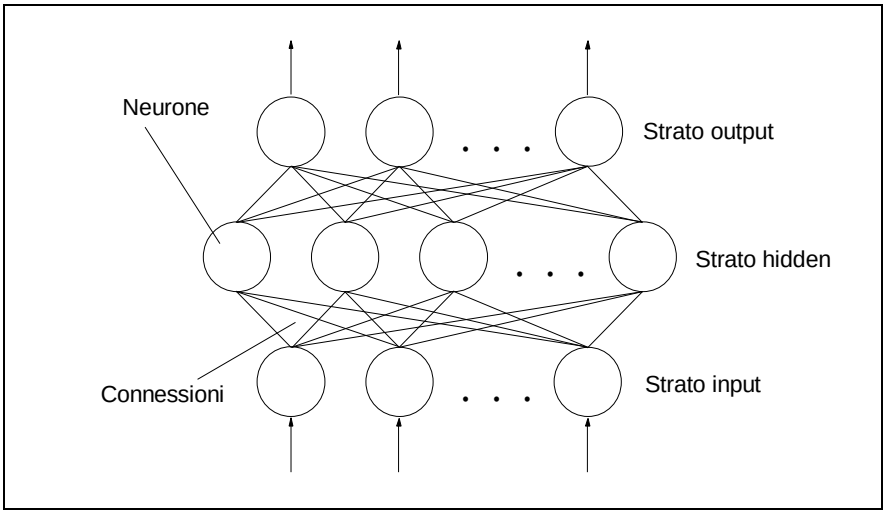

Fig. 2.3 - schema generico di una rete neurale.

Il funzionamento di una rete neurale può essere schematizzato in due fasi: la fase di apprendimento e quella di riconoscimento. Nella fase di apprendimento (***training***) la rete viene istruita su un campione di dati presi dall'insieme di quelli che dovranno poi essere elaborati; nella fase di riconoscimento (***testing***), che è poi quella di normale funzionamento, la rete è utilizzata per elaborare i dati in ingresso in base alla configurazione raggiunta nella fase precedente.

Per quanto riguarda le realizzazioni, pur possedendo le reti una struttura autonoma, generalmente si utilizzano simulazioni effettuate al computer in modo da permettere modifiche anche sostanziali in tempi brevi e con costi limitati. Stanno però nascendo i primi chip neurali che hanno performance notevolmente superiori a quelle di una simulazione ma che hanno finora avuto scarsissima diffusione dovuta soprattutto ai costi elevati e ad un'estrema rigidità strutturale.

2.4 Pattern recognition.

Il pattern recognition è attualmente l'area di più grande impiego delle reti neurali. Esso consiste nella classificazione di oggetti della più svariata natura in classi definite a priori o create automaticamente dall'applicazione in base alle somiglianze tra gli oggetti in input (si parla in questo caso di *clustering*). Il problema di decidere se una lettera è una *A* o una *B* è un problema di pattern recognition così come lo è anche decidere se un oggetto di un campo stellare è una stella o una galassia.

Per eseguire tramite computer compiti di classificazione, gli oggetti reali devono essere rappresentati in forma numerica e questo si fa effettuando, in modo opportuno, una modellazione della realtà che associ ad ogni oggetto un *pattern* (vettore di attributi numerici) che lo identifica. Questa prima fase prende il nome di *estrazione delle caratteristiche*. Le caratteristiche estratte possono risultare eccessive o ridondanti per una buona classificazione dell'oggetto, quindi si può pensare di ridurle in modo da sveltire il processo di classificazione. Questa operazione può essere effettuata manualmente o con tecniche automatiche quali la *feature selection* o la *principal component analisys* (vedi capitolo 5) ottenendo una traslazione dei pattern in un nuovo spazio di caratteristiche classificabile più semplicemente. Dopo questa ulteriore fase che prende il nome di *preprocessing* si passa finalmente alla classificazione vera e propria. Un *classificatore* può essere visto come una scatola nera capace di associare ogni pattern in input ad una determinata classe. Supponiamo, più formalmente, di dover classificare un pattern $x = (x_1 x_2 ... x_n)$ in una classe appartenente all'insieme $C = \{c_1, c_2, ..., c_k\}$ tramite il classificatore di figura 2.4: forniremo sulle linee di input il pattern *x* e il classificatore ci restituirà in output il vettore binario *y* costruito in questo modo:

$$y_i = \begin{cases} 1 \text{ se il pattern appartiene alla classe } c_i \\ 0 \text{ altrimenti.} \end{cases}$$

Le reti neurali possono essere efficacemente utilizzate come classificatori grazie alla loro capacità di apprendimento da esempi e di generalizzazione. L'idea è quella di far apprendere alla rete neurale (tramite appositi *algoritmi di training*) la corretta classificazione di un campione rappresentativo di pattern per poi far lavorare la stessa rete sull'insieme di tutti i pattern possibili. Distinguiamo a questo punto due tipi diversi di apprendimento: supervisionato e non supervisionato.

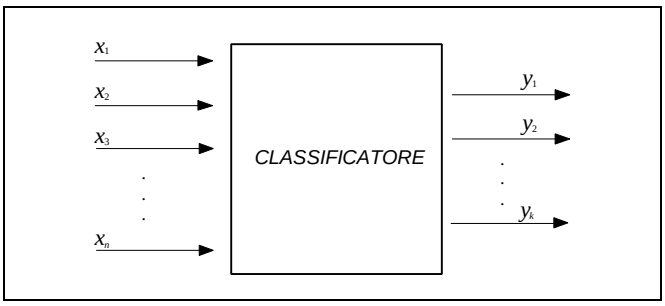

Fig. 2.4 - un classificatore generico.

Nell'***apprendimento supervisionato*** l'insieme dei pattern sul quale la rete deve apprendere (*training set*) è corredato da un insieme di etichette che mostrano la corretta classificazione di ogni pattern. In questo modo la rete effettua una regolazione della sua struttura e dei suoi parametri interni (pesi delle connessioni e soglie) finché non ottiene una corretta classificazione dei pattern di training. Date le sopraccitate capacità di generalizzazione, la rete funzionerà correttamente anche su pattern esterni ed indipendenti dal training set purché il training set stesso sia sufficientemente rappresentativo.

Nell'***apprendimento non supervisionato*** non è associabile al training set un insieme di etichette. Questo può avvenire per vari motivi: le classi di appartenenza possono essere semplicemente sconosciute e non ottenibili manualmente o ottenibili solo in maniera imprecisa o lenta o, ancora, la conoscenza a-priori potrebbe essere ambigua (uno stesso pattern potrebbe essere etichettato in maniera diversa da diversi esperti). In questo tipo di apprendimento, la rete tenta di organizzare i pattern del training set in sottoinsiemi detti *cluster* utilizzando opportune misure di similarità in modo tale che tutti gli oggetti appartenenti ad uno stesso cluster sono tra loro il più possibile simili mentre gli oggetti appartenenti a cluster distinti siano tra loro il più possibile diversi. Successivamente è necessario utilizzare la conoscenza a-priori di un esperto per etichettare i cluster ottenuti al passo precedente al fine di rendere utilizzabile il classificatore.

Questi due diversi approcci all'apprendimento danno luogo a diversi tipi di reti neurali che saranno analizzate in dettaglio nei prossimi capitoli prestando particolare attenzione alle tecniche effettivamente utilizzate nel presente lavoro di tesi.

Riferimenti.

R. Beale, T. Jackson “Neural Computing: An Introduction” *Institute of Physics Publishing Bristol and Philadelphia*, 1980.

A. De Luca, M. Ricciardi “Introduzione alla Cibernetica” *Franco Angeli Editore*, 1986.

L. Macera “Reti Neurali e Pattern Recognition - finalmente un cervello elettronico?” in *MC-Microcomputer*, no. 102, pp. 210-213, 1990.

A. Mazzetti “Intelligenza e Vita” *Apogeo-Editrice di Informatica*, 1994.

CAPITOLO 3

Apprendimento Supervisionato

3.1 Introduzione.

Descriveremo in questo capitolo diversi modelli di *reti neurali ad apprendimento supervisionato* con particolare riferimento a quelli utilizzati in questo lavoro di tesi. Come abbiamo già accennato nel paragrafo 2.4, questi modelli necessitano di *training set* costituiti da coppie del tipo *pattern-etichetta*. In altri termini, per ogni esempio fornito, deve essere specificato anche la classe di appartenenza in modo che la rete possa modificare la sua struttura interna fino a minimizzare l'errore tra output atteso e output effettivo per ogni esempio. La rete sarà quindi in grado, dopo questo processo di apprendimento, di classificare correttamente il *training set* e, se questo è una buona rappresentazione dell'universo, anche *pattern* esterni ad esso.

Il modello più semplice di rete neurale ad apprendimento supervisionato è il ***perceptron***, costituito da un solo neurone con *n* input. Le capacità di calcolo di questo modello sono molto limitate, la sua importanza risiede soprattutto nel fatto di essere un punto di partenza per modelli più complessi e potenti. La sua architettura è quella di figura 2.2 tranne che l'output è definito come segue:

$$y = H\left(\sum_{i=1}^{n} w_i x_i - q\right) \text{dove } H(z) = \begin{cases} 1 \text{ se } z > 0 \\ 0 \text{ se } z \le 0 \end{cases}$$

(la H è detta funzione di attivazione di Heaviside). La precedente espressione può essere semplificata introducendo un input fittizio (*bias term*) $x_0 = 1$ tale che $w_0 = -q$ in modo da ottenere:

$$y = H\left(\sum_{i=0}^{n} w_i x_i\right).$$

La rete apprende alterando i pesi in modo tale da rinforzare le decisioni corrette e scoraggiare quelle scorrette. Ad ogni iterazione si presenta alla rete un nuovo *pattern* del *training set* sul quale la rete calcola il suo output. Si procede poi alla modifica dei pesi con l'espressione $w_i^{(t+1)} = w_i^{(t)} + h(c - y)x_i$ dove c è l'etichetta di classe (ovvero

l'output atteso) e η è un *gain term* (tale che $0 < h \leq 1$) che regola la velocità dell'apprendimento.

Il limite principale di questo modello è che riesce a classificare correttamente l'input solo se le classi sono linearmente separabili (vedi figura 3.1a). Spesso, però, la divisione tra la classi è più complessa. Un esempio classico è il problema dello XOR (vedi figura 3.1b) dove non è possibile trovare una sola linea separatrice per le due classi e quindi il *perceptron* fallisce. Per ovviare a questi problemi è necessario ricorrere a strutture più complesse organizzate su più strati. Descriveremo questi nuovi modelli nei prossimi paragrafi.

3.2 Il Multi-Layer Perceptron.

Abbiamo visto nel precedente paragrafo come un neurone sia in grado di individuare due semispazi; vedremo ora come, intersecando tali semispazi, si possono individuare regioni arbitrarie. Bisogna, prima di tutto, strutturare la rete su più livelli totalmente connessi, tali cioè che l'output di un neurone dell'*i*-esimo strato sia l'input di tutti i neuroni allo strato *i*+1. In questo modo, se con uno strato otteniamo solo semipiani (figura 3.2a), con due strati otterremo intersezioni di semipiani ovvero regioni convesse (figura 3.2b) e con 3 strati intersezioni di regioni convesse ovvero figure arbitrarie (figura 3.2c) che ci consentiranno di delimitare una qualsiasi classe.

Questa è la filosofia delle reti *Multi-Layer Perceptron* (da ora in poi MLP) che sono

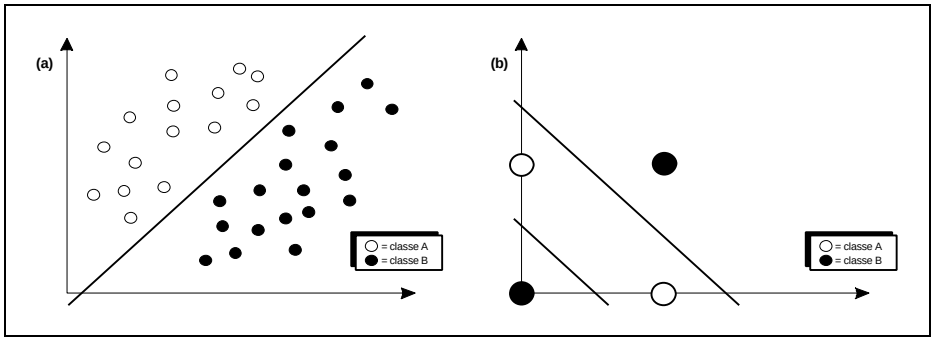

Fig. 3.1 - separazioni corrette e scorrette di classi operate da un perceptron.

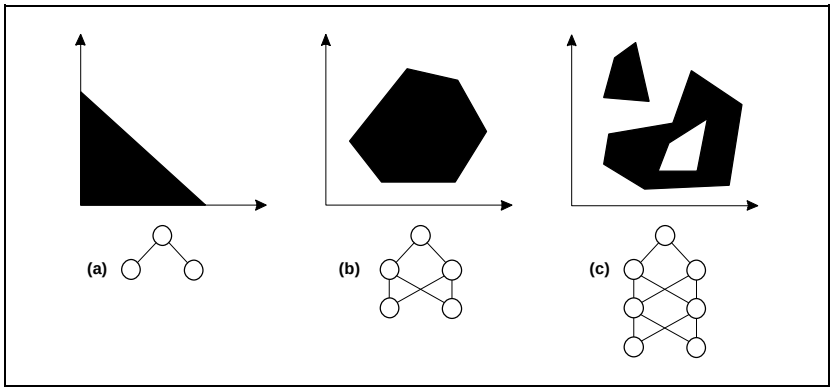

Fig. 3.2 - regioni individuabili con MLP a 1, 2 e 3 strati.

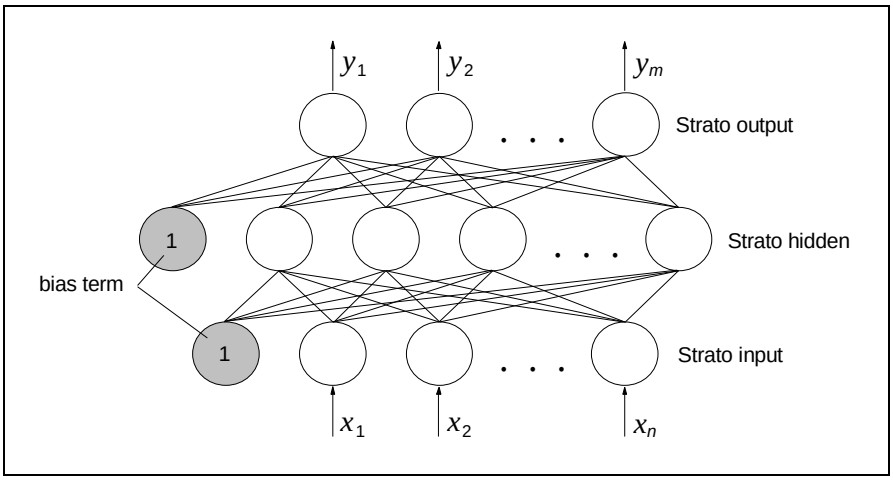

Fig. 3.3 - topologia di una MLP.

formate da uno strato di neuroni input che non fanno nessuna elaborazione e il cui compito è solo quello di far arrivare l'input a tutti i neuroni dello strato successivo; da uno o più livelli interni detti *hidden*; e da uno strato di output. Ogni strato è totalmente connesso al successivo, ogni connessione è caratterizzata da un peso ed ogni neurone, tranne quelli che appartengono allo strato di input, ha un input in più costituito dal *bias term*. La topologia della rete è mostrata in figura 3.3.

Utilizzando questo tipo di reti è possibile individuare qualsiasi tipo di regione ma, di contro, il processo di apprendimento risulta molto più laborioso rispetto a quello del *perceptron*. Se si volesse utilizzare la stessa regola di apprendimento del *perceptron* non si avrebbe alcun modo di determinare, in caso di errore, quale delle unità interne ne è responsabile e non si saprebbe, di conseguenza, quale peso modificare per correggere il comportamento della rete. Una soluzione a questo problema è però possibile se si utilizza, al posto della funzione di Heaviside (figura 3.4a), una funzione di attivazione differenziabile come, per esempio, la sigmoidale di figura 3.4b. In questo caso, i valori di attivazione delle unità di output diventano funzioni differenziabili dei valori di input e dei pesi ovvero (nel caso di un solo neurone):

$$y = f\left(\sum_{i=0}^{n} w_i x_i\right) \tag{3.1}$$

Se si definisce, inoltre, una funzione errore come la somma dei quadrati degli scarti tra output attesi e output effettivi si trova che essa stessa è una funzione differenziabile degli output e quindi anche dei pesi. L'apprendimento basato su questo principio è chiamato *backpropagation* in virtù del fatto che l'errore viene propagato all'indietro nella rete a partire dai neuroni di output. Vediamo in dettaglio il suo funzionamento.

Si consideri un generico *pattern* input *n*-dimensionale *p* e il corrispondente *target* (vettore output che ci si aspetta dalla rete) c_p e si definisca la funzione di errore per il *pattern p* come segue:

$$E_p = \frac{1}{2}\sum_i \left(c_{pi} - y_{pi}\right)^2 \tag{3.2}$$

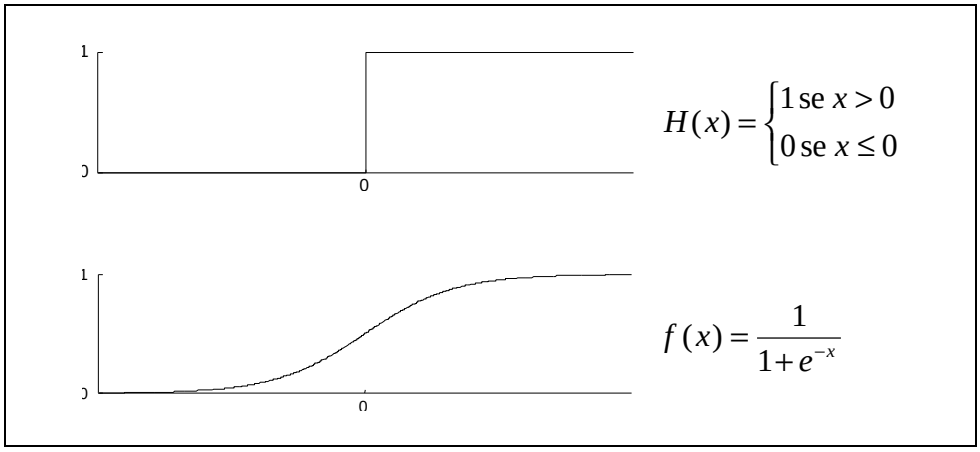

Fig. 3.4 - esempi di funzioni di attivazione.

dove y_{pi} è il responso dell'*i*-esimo neurone della rete. L'algoritmo di apprendimento deve aggiornare i pesi in modo tale da minimizzare questa funzione di errore seguendo quello che si indica come *gradiente discendente*. Da come è stata definita la rete risulta che l'input e l'output del generico neurone *j* sono dati rispettivamente da:

$$net_{pj} = \sum_i w_{ij} y_{pi} \quad \text{e} \quad y_{pj} = f(net_{pj}) \tag{3.3}$$

dove *f* è la funzione sigmoidale. La derivata della funzione d'errore rispetto ai pesi, utilizzando la regola della catena, può essere scritta come:

$$\frac{\partial E_p}{\partial w_{ij}} = \frac{\partial E_p}{\partial net_{pj}} \frac{\partial net_{pj}}{\partial w_{ij}} \tag{3.4}$$

e, prestando attenzione al secondo termine, si può effettuare la seguente sostituzione:

$$\frac{\partial net_{pj}}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \sum_k w_{kj} y_{pk} = \sum_k \frac{\partial w_{kj}}{\partial w_{ij}} y_{pk} = y_{pi} \tag{3.5}$$

dove l'ultimo passaggio deriva dal fatto che $\partial w_{kj} / \partial w_{ij} = 0$ eccetto per $k=i$ dove vale 1.

Indicando il primo fattore del secondo termine della 3.4 con $-\delta_{pj}$, è possibile scrivere:

$$-\frac{\partial E_p}{\partial w_{ij}} = \delta_{pj} y_{pi}. \tag{3.6}$$

Per diminuire il valore di E_p si deve quindi variare il peso w_{ij} proporzionalmente a $\delta_{pj} y_{pi}$ ottenendo una regola di variazione del tipo:

$$w_{ij} = w_{ij} + \Delta w_{ij} \text{ dove } \Delta w_{ij} = \eta \delta_{pj} y_{pi}. \tag{3.7}$$

Vediamo ora quale è il valore di δ_{pj} per ogni neurone. Utilizzando la solita regola della catena si ha:

$$\delta_{pj} = -\frac{\partial E_p}{\partial net_{pj}} = \frac{\partial E_p}{\partial y_{pj}} \frac{\partial y_{pj}}{\partial net_{pj}} \tag{3.8}$$

dove il secondo fattore del secondo termine si può riscrivere come:

$$\frac{\partial y_{pj}}{\partial net_{pj}} = f'(net_{pj}) \tag{3.9}$$

mentre, per il primo fattore si devono distinguere due casi: se *j* è un neurone output:

$$\frac{\partial E_p}{\partial y_{pj}} = -(c_{pj} - y_{pj}) \quad \text{e quindi} \quad \delta_{pj} = (c_{pj} - y_{pj}) f'(net_{pj}); \tag{3.10}$$

se *j* non è un neurone output si ha invece:

$$\frac{\partial E_p}{\partial y_{pj}} = \sum_k \frac{\partial E_p}{\partial net_{pk}} \frac{\partial net_{pk}}{\partial y_{pj}} = -\sum_k \delta_{pk} w_{jk} \quad \text{e quindi} \quad \delta_{pj} = f'(net_{pj}) \sum_k \delta_{pk} w_{jk}. \tag{3.11}$$

Nel caso in cui la funzione di attivazione del neurone è sigmoidale, il discorso relativo al calcolo della derivata si semplifica molto. Infatti: $f'(net_{pj}) = y_{pj}(1 - y_{pj})$ e:

$$\delta_{pj} = (c_{pj} - y_{pj}) y_{pj}(1 - y_{pj}) \quad \text{oppure} \quad \delta_{pj} = (c_{pj} - y_{pj}) \sum_k \delta_{pk} w_{jk} \tag{3.12}$$

a seconda se il neurone *j* è o non è un neurone output. La figura 3.5 mostra l'algoritmo completo di apprendimento di una rete MLP standard.

Come si è visto, l'apprendimento basato sul principio della *backpropagation* cerca di adattare i pesi della rete in modo da minimizzare la funzione errore $E(w)$. Per le reti aventi un singolo livello, la funzione di errore è quadratica rispetto ai pesi e avrà, di conseguenza, una forma parabolica multidimensionale con un solo punto di minimo. La soluzione del problema è in questo caso banale. Purtroppo, per reti più generali, la funzione errore sarà una funzione generica non lineare dei pesi che avrà più punti di minimo in cui il gradiente dell'errore è nullo.

Si consideri una **MLP** con *m* output e *n* input e si indichi con $w_{ij}^{(t)}$ il peso tra l'*i*-esimo e il *j*-esimo neurone al tempo *t*.

1. Si inizializzino tutti i pesi $w_{ij}^{(0)}$ con valori piccoli e casuali.

2. Si presenti un nuovo *pattern* $p = (x_1, \ldots, x_n)$ insieme al target $c_p = (c_{p1}, \ldots, c_{pm})$ che si aspetta in output dalla rete.

3. Si calcoli l'output di ogni neurone *j* della rete (strato per strato) in questo modo:

$$y_{pj} = f\left(\sum_i w_{ij} x_{pi}\right)$$

tranne che per i neuroni input per i quali l'output è uguale all'input.

4. Si adattino i pesi partendo dall'ultimo strato e andando a ritroso utilizzando l'equazione:

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} + \eta\, \delta_{pj}\, y_{pj}$$

dove η è il *gain term* (tale che $0 < \eta \leq 1$) e δ_{pj} è il termine d'errore definito alla 3.12.

5. Si ripeta il procedimento (tornando al passo **2**) per tutti i *pattern* del *training set*.

Fig. 3.5 - algoritmo di apprendimento di una MLP standard.

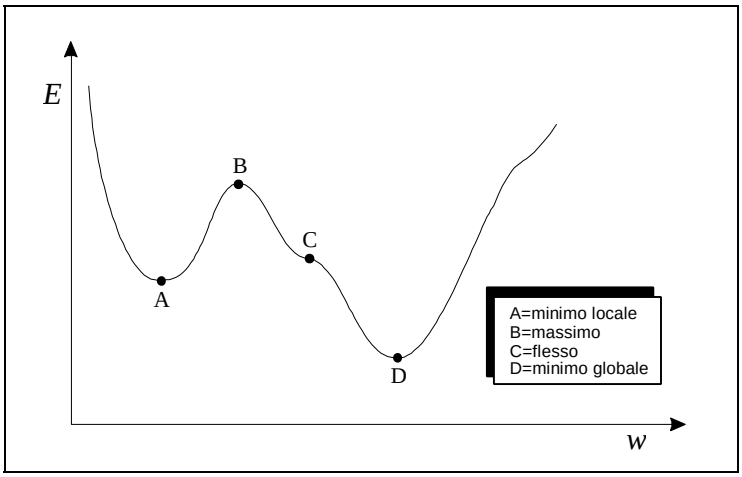

Fig. 3.6 - esempio di una funzione di errore.

In questo caso bisogna distinguere tra minimo assoluto e minimi locali. Esistono inoltre altri punti dove il gradiente dell'errore è nullo chiamati punti di flesso e punti di massimo. Un esempio schematico di funzione errore vista rispetto a un singolo parametro *w* è mostrato in figura 3.6.

L'algoritmo di apprendimento della MLP adatta i pesi addizionando a questi un vettore di incremento ad ogni passo. Lo svantaggio principale di questa tecnica è che, se la rete si dovesse posizionare in un punto di minimo locale della funzione errore, vi rimarrebbe per sempre e non ci sarebbe alcun sistema per scavalcarlo. Ogni direzione, da quel punto, porta ad un valore più alto dell'errore, la rete conclude quindi di aver raggiunto la configurazione ottima e l'algoritmo termina.

Un'ulteriore critica mossa alla MLP è che essa richiede troppi cicli (quindi ripetizioni dei calcoli della funzione errore) prima di giungere ad una soluzione stabile. Il metodo del gradiente discendente è purtroppo intrinsecamente lento nel convergere in un panorama complesso quale quello fornito dalla funzione errore in un caso non banale. Per nostra fortuna esistono metodi alternativi per aggirare questi ostacoli.

3.3 Miglioramenti alla tecnica del gradiente discendente.

Esistono in letteratura due versioni diverse dell'Algoritmo del Gradiente Discendente (d'ora in poi GDA): la *sequenziale* e la *batch*. Nella ***versione sequenziale*** (che abbiamo descritto nel paragrafo precedente), i pesi vengono modificati valutando il gradiente dell'errore ogni volta che viene presentato un *pattern* alla rete. Se si indica con *w* il vettore dei pesi della rete e con $\nabla E_p|_w$ il gradiente della funzione di errore relativa al *pattern p* valutato nel punto *w,* si può riscrivere la 3.7 in forma vettoriale come:

$$w^{(t+1)} = w^{(t)} + \Delta w^{(t)} \quad \text{dove} \quad \Delta w^{(t)} = -h \nabla E_p|_{w^{(t)}} \tag{3.13}$$

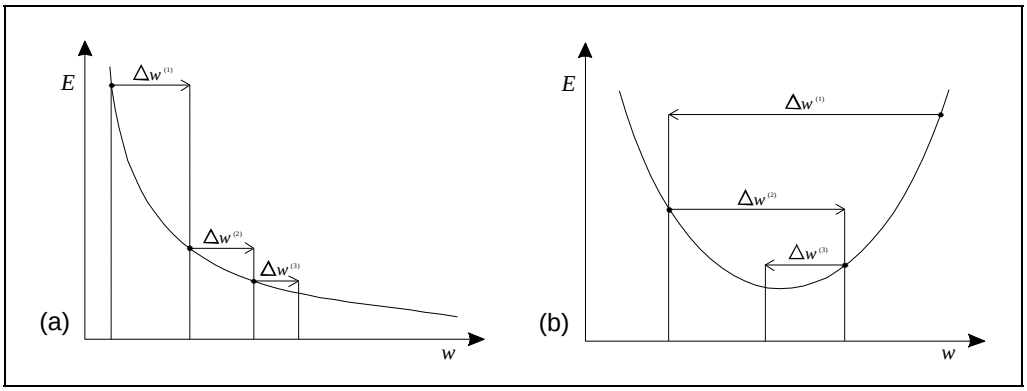

Fig. 3.7 - variazione dei pesi su differenti funzioni di errore.

e i *pattern* del *training set* possono essere considerati o in sequenza o scelti in modo casuale. Nella ***versione batch***, invece, l'algoritmo aggiorna i pesi solo dopo aver presentato l'intero *training set* in modo che la variazione sia pari a:

$$\Delta w^{(t)} = -h\nabla E|_{w^{(t)}} \quad \text{dove} \quad \nabla E = \sum_p \nabla E_p . \tag{3.14}$$

Tra i due approcci è preferibile il primo se si riscontra un alto grado di ridondanza nell'informazione del *training set*, il secondo in caso contrario. Entrambe le versioni richiedono che si definisca a priori il ***learning rate*** h. È questo un compito molto delicato dato che, per h troppo grande, l'algoritmo potrebbe non convergere, mentre per h troppo piccolo, la ricerca del minimo potrebbe risultare estremamente lenta. Una soluzione migliore è di diminuire h ad ogni passo (ad esempio ponendo $h = 1/t$) ottenendo così un algoritmo di apprendimento più veloce. Un ulteriore vantaggio deriva dal fatto che se il termine di guadagno viene inizializzato con un valore relativamente grande, si ottengono inizialmente grandi variazioni dei pesi (quindi grossi spostamenti sulla funzione errore) che scavalcano i minimi locali.

Una tecnica molto semplice per migliorare l'apprendimento è quella di sommare un ***termine momento*** alla formula del GDA in modo che (nel caso batch) la variazione dei pesi diventa:

$$\Delta w^{(t)} = -h\nabla E^{(t)} + m\Delta w^{(t-1)} \quad \text{dove} \quad \nabla E^{(t)} = \nabla E|_{w^{(t)}} \tag{3.15}$$

dove m è il *parametro di momento*. Per capire l'effetto del termine momento, consideriamo innanzitutto i valori successivi dei pesi in corrispondenza di una regione dell'errore con lieve pendenza come in figura 3.7a. Se si approssima il valore del gradiente, si può applicare la 3.15 iterativamente ad una lunga serie di modifiche dei pesi e sommare il risultato della serie aritmetica ottenendo:

$$\Delta w = -h\nabla E(1 + m + m^2 + ...) = -\frac{h}{1-m}\nabla E \tag{3.16}$$

dove si può notare che il risultato del termine momento è quello di incrementare l'effettivo *learning rate* da h a $h/(1-m)$. Ciò comporta una velocizzazione dell'algoritmo. D'altra parte, in una regione a grande curvatura (vedi figura 3.7b) nella quale il gradiente discendente è oscillatorio, successivi contributi del termine momento

Si consideri una **MLP** e si indichi con $w^{(t)}$ il vettore di pesi al tempo *t*.

1. Si inizializzi $w^{(0)}$ con valori piccoli e casuali, si settino m, h, r, s e e, si ponga t=0.

2. Si presenti tutto il *training set* e si calcoli $E(w^{(t)})$ ovvero la funzione di errore rispetto alla configurazione attuale dei pesi.

3. Si calcoli $E(w')$ dove $w' = w^{(t)} - h\nabla E^{(t)} + m\Delta w^{(t-1)}$; se $E(w') < E(w^{(t)})$ allora $w^{(t+1)} = w'$ e $h = hr$ e si vada al passo **4**; altrimenti $h = hs$ e si ritorni al passo **3**.

4. Se $E(w^{(t+1)}) > e$ allora $t = t+1$ e si ritorni al passo **2** altrimenti **STOP**.

Fig. 3.8 - GDA batch con bold driver e termine momento.

tendono a cancellarsi e l'effettivo *learning rate* sarà h. L'adozione del termine momento non rende quindi l'algoritmo molto più efficiente e, inoltre, introduce un altro parametro il cui valore deve essere scelto a priori.

Un altro inconveniente sia del termine momento che del *learning rate* è che il valore ottimo di questi parametri dipende dal particolare problema e tipicamente varia durante tutto il *training*. Sarebbe quindi auspicabile una procedura che calcoli i parametri ottimali in modo automatico. A tale proposito si introduce la tecnica del ***bold driver***. Per capirne il funzionamento, si consideri innanzitutto l'apprendimento senza il termine momento. L'idea è quella di controllare che la funzione errore, dopo un aggiornamento dei pesi provvisorio, stia effettivamente decrescendo. Se l'errore cresce, allora il minimo è stato scavalcato e, quindi, il *learning rate* è troppo grande. In questo caso la variazione dei pesi viene annullata ed il *learning rate* decrementato. Si ripete questo procedimento fino a che non si ottiene un effettivo decremento della funzione errore, nel qual caso la nuova variazione dei pesi viene confermata. Se, al contrario, l'errore decresce, allora l'aggiornamento dei pesi viene confermato ed il *learning rate* incrementato per un successivo passo. L'algoritmo segue la seguente regola:

$$h_{new} = \begin{cases} r h_{old} \text{ se } \Delta E < 0 \\ s h_{old} \text{ se } \Delta E > 0 \end{cases} \qquad (3.17)$$

dove il parametro r è scelto leggermente più grande dell'unità (tipicamente $r = 1.1$), mentre s viene scelto abbastanza più piccolo dell'unità (tipicamente $s = 0.5$). Se si include anche il termine momento, si ottiene l'algoritmo di figura 3.8.

3.4 Line search e gradienti coniugati.

Utilizzando il GDA standard, la direzione di ogni passo di adattamento è calcolata tramite il gradiente discendente dell'errore mentre la lunghezza è determinata dal *learning rate*. Un approccio più sofisticato potrebbe essere quello di muoversi verso la direzione negativa del gradiente (*search direction*) non di una lunghezza fissa, ma fino a raggiungere il minimo della funzione in quella direzione. Ciò è possibile calcolando il gradiente discendente e studiandolo al variare del *learning rate*. Si supponga che al

passo t il corrente vettore dei pesi sia $w^{(t)}$ e si consideri una direzione di ricerca $d^{(t)} = -\nabla E^{(t)}$. Se scegliamo il parametro l in modo da minimizzare:

$$E(l) = E(w^{(t)} + ld^{(t)}). \tag{3.18}$$

Il nuovo vettore di pesi può essere allora scritto come:

$$w^{(t+1)} = w^{(t)} + ld^{(t)} \tag{3.19}$$

Il problema del ***line search*** si traduce quindi in un problema di minimizzazione in una dimensione. Un soluzione molto semplice consiste nel muoversi su $E(l)$ variando λ di piccoli intervalli, valutare la funzione errore in ogni nuova posizione e fermarsi quando l'errore comincia a crescere (vedi figura 3.9a). Esistono molti altri sistemi per risolvere il problema in modo più efficiente. Uno di questi (descritto in figura 3.9b) consiste nel considerare tre valori *a*, *b*, *c* del tasso di apprendimento in modo da avere $a<b<c$, $E(a) > E(b)$ e $E(c) > E(b)$ e calcolare la parabola passante per questi punti. Il minimo *d* della parabola è una buona approssimazione del minimo di $E(l)$ e può essere ulteriormente avvicinato considerando la parabola passante per *d* e gli altri due punti (nell'esempio *b* e *c*) con i valori di errore più bassi.

Si è finora supposto che la direzione di ricerca ottimale per il metodo del *line search* sia data, ad ogni passo, dal gradiente negativo. Ciò non è sempre vero dato che, se si minimizza rispetto alla direzione del gradiente negativo, la successiva direzione di ricerca (il nuovo gradiente) sarà ortogonale alla precedente. Basta infatti notare che, quando il *line search* trova il minimo allora, dalla 3.18 si ottiene che:

$$\frac{\partial E}{\partial l}\left(w^{(t)} + ld^{(t)}\right) = 0 \text{ e quindi } g^{(t+1)T} d^{(t)} = 0 \tag{3.20}$$

dove $g^{(t+1)} \equiv \nabla E^{(t+1)}$. Scegliendo successive direzioni uguali al gradiente negativo, si possono quindi ottenere delle oscillazioni sulla funzione di errore che rallentano il processo di convergenza.

La soluzione a questo problema si ottiene scegliendo direzioni successive così che, ad ogni passo dell'algoritmo, la componente del gradiente parallela alla precedente direzione di ricerca, che è 0, rimanga inalterata. Supponiamo di aver già minimizzato rispetto alla direzione $d^{(t)}$ partendo dal punto $w^{(t)}$ ed arrivando al punto $w^{(t+1)}$. Nel

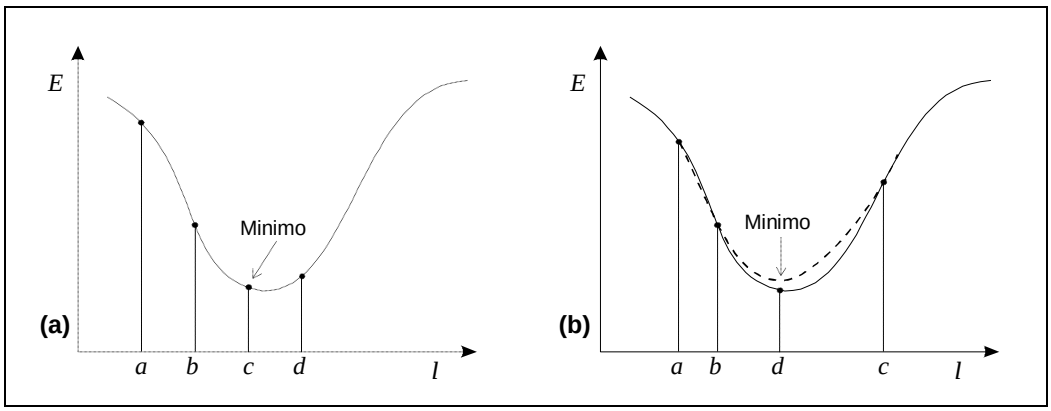

Fig. 3.9 - metodo degli intervalli e metodo delle parabole per la ricerca del minimo.

punto $w^{(t+1)}$ vale quindi (dalla 3.20):

$$g\left(w^{(t+1)}\right)^T d^{(t)} = 0\,. \tag{3.21}$$

Scegliendo ora $d^{(t+1)}$ in modo tale da conservare la componente del gradiente parallela a $d^{(t)}$ uguale a 0 ovvero tale che:

$$g\left(w^{(t+1)}\right)^T d^{(t)} = 0 \tag{3.22}$$

è possibile costruire una sequenza di direzioni *d* in modo tale che ogni direzione è *coniugata* alle precedenti sulla dimensione |*w*| dello spazio di ricerca.

E' possibile dimostrare, in presenza di una funzione di errore quadratica, che un algoritmo che implementa una tale tecnica ha un adattamento pesi del tipo:

$$w^{(t+1)} = w^{(t)} + a^{(t)} d^{(t)} \quad \text{dove} \quad a^{(t)} = -\frac{d^{(t)\mathrm{T}} g^{(t)}}{d^{(t)\mathrm{T}} H d^{(t)}}. \tag{3.23}$$

Inoltre, *d* si può ottenere la prima volta dal gradiente negativo e poi come combinazione lineare del gradiente corrente e delle precedenti direzioni di ricerca:

$$d^{(t+1)} = -g^{(t+1)} + b^{(t)} d^{(t)} \quad \text{dove} \quad b^{(t)} = \frac{g^{(t+1)\mathrm{T}} H d^{(t)}}{d^{(t)\mathrm{T}} H d^{(t)}}. \tag{3.24}$$

L'algoritmo appena descritto trova il minimo di una funzione di errore quadratica in al massimo |*w*| passi. Bisogna però notare che il costo computazionale di ogni passo è piuttosto elevato dato che, per determinare i valori di α e β, si fa riferimento alla *matrice Hessiana H* il cui calcolo è molto costoso. Fortunatamente i coefficienti α e β possono essere ottenuti anche da espressioni che non utilizzano la *matrice Hessiana* in modo esplicito. Il termine β può essere calcolato in uno dei seguenti modi:

1. espressione di *Hestenes-Sitefel*: $$b^{(t)} = \frac{g^{(t+1)\mathrm{T}}(g^{(t+1)} - g^{(t)})}{d^{(t)\mathrm{T}}(g^{(t+1)} - g^{(t)})}; \tag{3.25}$$

2. espressione di *Polak-Ribiere*: $$b^{(t)} = \frac{g^{(t+1)\mathrm{T}}(g^{(t+1)} - g^{(t)})}{g^{(t)\mathrm{T}} g^{(t)}}; \tag{3.26}$$

3. espressione di *Fletcher-Reeves*: $$b^{(t)} = \frac{g^{(t+1)\mathrm{T}} g^{(t+1)}}{g^{(t)\mathrm{T}} g^{(t)}}. \tag{3.27}$$

Queste tre espressioni sono equivalenti se la funzione di errore è quadratica, in caso contrario assumono valori differenti ma la 3.26 ottiene, in genere, risultati migliori. Ciò accade probabilmente a causa del fatto che, se l'algoritmo procede lentamente (in modo tale che i successivi gradienti siano molto simili) la 3.26 produce valori piccoli per β così che la direzione di ricerca tende ad essere riportata alla direzione negativa del gradiente, il che equivale ad un *restart* della procedura.

Per quanto riguarda il parametro α, il suo valore può essere ricavato invece che dalla 3.23, utilizzando direttamente il *line search.*

Il metodo dei *gradienti coniugati* riduce i passi per minimizzare l'errore ad un massimo di |*w*| dato che ci possono essere al massimo |*w*| direzioni coniugate in uno spazio |*w*|-dimensionale. In pratica, però, l'algoritmo è più lento dato che, durante il

Si consideri una **MLP** e si indichi con $w^{(t)}$ il vettore di pesi al tempo *t*.

1. Si inizializzi $w^{(0)}$ con valori piccoli e casuali, si setti la costante *e* e si ponga *t*=0.

2. Si presenti tutto il *training set* e si calcoli $E(w^{(t)})$ ovvero la funzione di errore rispetto alla configurazione attuale dei pesi.

3. Se *t*=0 o *t* è un multiplo di |*w*| allora $d^{(t)} = -\nabla E^{(t)}$ altrimenti $d^{(t)} = -\nabla E^{(t)} + b^{(t-1)} d^{(t-1)}$ dove $b^{(t-1)}$ assume il valore definito dalla **3.26** con $g^{(t)} = -\nabla E^{(t)}$.

4. Si calcoli $w^{(t+1)} = w^{(t)} - a d^{(t)}$ dove α è ottenuto dal line search.

5. Se $E(w^{(t+1)}) > e$ allora $t = t + 1$ e si ritorni al passo **2** altrimenti **STOP**.

Fig. 3.10 - algoritmo dei gradienti coniugati.

processo di apprendimento, la proprietà coniugata delle direzioni di ricerca tende a deteriorarsi. È utile, per sopperire a questa carenza, far ripartire l'algoritmo dopo |*w*| passi resettando la direzione di ricerca con la direzione negativa del gradiente. La figura 3.10 mostra l'algoritmo dei *gradienti coniugati* come una particolare forma del GDA nel quale il *learning rate* η è determinato dal *line search* mentre il *termine momento* μ è ottenuto dal parametro β.

3.5 Scaled Conjugate Gradients.

Nell'algoritmo dei *gradienti coniugati*, il parametro α è determinato tramite il *line search* per evitare il calcolo della matrice Hessiana. L'algoritmo è molto sensibile al parametro α quindi il *line search* deve essere molto accurato e ciò implica un tempo di calcolo molto lungo. L'algoritmo che vedremo in questo paragrafo utilizza un approccio diverso che evita l'utilizzo del *line search*. Nell'espressione 3.23, la *matrice Hessiana* è moltiplicata per il vettore *d* ed è possibile dimostrare che, per una rete MLP, il prodotto tra la matrice *H* ed un vettore arbitrario, può essere valutato in $O(|w|)$ passi (per ogni *pattern* di *training*) usando la tecnica dell'operatore $R\{\cdot\}$. Questo approccio è però inefficiente in presenza di funzioni errore non quadratiche dato che la *matrice Hessiana* potrebbe non essere positiva. In questo caso il denominatore della 3.23 potrebbe essere negativo portando a valori peso che incrementano la funzione errore. Il problema può essere risolto imponendo che la matrice *H* sia sempre positiva addiziondole un multiplo della matrice identità lI dove *I* è la matrice identità e $l > 0$ è un *coefficiente di scaling*. In questo modo si ottiene la seguente espressione per α:

$$a^{(t)} = -\frac{d^{(t)\mathrm{T}} g^{(t)}}{d^{(t)\mathrm{T}} H d^{(t)} + l^{(t)} ||d^{(t)}||^2} \tag{3.28}$$

dove l'indice (*t*) riflette il fatto che il valore ottimo per λ varia per ogni iterazione.

Questa tecnica è conosciuta come *model trust region* perché il modello è effettivamente valido solo entro piccole regioni intorno al corrente punto di ricerca. La

grandezza della *trust region* è determinata dal parametro λ. Bisogna ora trovare un metodo per scegliere un appropriato valore per λ. Con λ=0 i pesi si muovono verso un minimo solo se la funzione errore è quadratica ed il denominatore è positivo. Se una di queste condizioni non è verificata allora λ deve essere incrementato. Consideriamo il caso in cui la matrice *H* non è positiva. Se indichiamo con $d^{(t)}$ il denominatore della 3.28 allora, siccome $d^{(t)} < 0$, bisogna incrementare λ finché non risulta che $d^{(t)} > 0$. Se indichiamo con $\bar{l}$ il valore aggiornato di λ allora:

$$\bar{d}^{(t)} = d^{(t)} + (\bar{l}^{(t)} - l^{(t)})\|d^{(t)}\|^2 \tag{3.29}$$

che sarà positivo se:

$$\bar{l} = 2\left(l^{(t)} - \frac{d^{(t)}}{\|d^{(t)}\|^2} \right) \tag{3.30}$$

Sostituendo si ottiene:

$$\bar{d}^{(t)} = -d^{(t)} + l^{(t)}\|d^{(t)}\|^2 = -d^{(t)\mathrm{T}} H^{(t)} d^{(t)\mathrm{T}} \tag{3.31}$$

che è positivo e si può utilizzare come denominatore della 3.28.

Bisogna ora considerare un'approssimazione quadratica della funzione errore nella regione individuata dal vettore peso corrente: se la funzione reale è simile a questa approssimazione, il valore di λ sarà ridotto; in caso contrario λ verrà incrementato. Il confronto si può effettuare valutando il parametro:

$$\Delta^{(t)} = \frac{E(w^{(t)}) - E(w^{(t)} + a^{(t)} d^{(t)})}{E(w^{(t)}) - E_Q(w^{(t)} + a^{(t)} d^{(t)})} \tag{3.32}$$

Dove $E_Q(w)$ è l'approssimazione quadratica della funzione errore attorno a *w* data da:

$$E_Q(w^{(t)} + a^{(t)} d^{(t)}) = E(w^{(t)}) + a^{(t)} d^{(t)\mathrm{T}} g^{(t)} + \frac{1}{2} a^{(t)2} d^{(t)\mathrm{T}} H^{(t)} d^{(t)} \tag{3.33}$$

Effettuando semplici sostituzioni si ottiene dalla 3.32:

$$\Delta^{(t)} = \frac{2\left[E(w^{(t)}) - E(w^{(t)} + a^{(t)} d^{(t)})\right]}{a^{(t)} d^{(t)\mathrm{T}} g^{(t)}} \tag{3.34}$$

da cui si deduce che il valore di λ può essere modificato usando la seguente regola:

$$l^{(t+1)} = \begin{cases} l^{(t)}/2 & \text{se } \Delta^{(t)} > 0.75; \\ 4\, l^{(t)} & \text{se } \Delta^{(t)} < 0.25; \\ l^{(t)} & \text{altrimenti}. \end{cases} \tag{3.35}$$

Si noti che con Δ<0 i pesi non vengono aggiornati dato che il passo porterebbe ad un incremento nell'errore. In questo caso si incrementa il valore di λ e si rivaluta Δ. Se otteniamo un decremento della funzione errore per un λ sufficiente, l'algoritmo si muove con un piccolo passo verso il gradiente negativo. La figura 3.11 mostra l'algoritmo completo *Scaled Conjugate Gradients*.

Si consideri una **MLP** e si indichi con $w^{(t)}$ il vettore di pesi al tempo *t*.

1. Si inizializzi $w^{(0)}$ con valori piccoli e casuali, si setti la costante *e* e si ponga *t*=0.

2. Si presenti tutto il *training set* e si calcoli $E(w^{(t)})$ ovvero la funzione di errore rispetto alla configurazione attuale dei pesi.

3. Se *t*=0 o *t* è un multiplo di |*w*| allora $d^{(t)} = -\nabla E^{(t)}$ altrimenti $d^{(t)} = -\nabla E^{(t)} + b^{(t-1)} d^{(t-1)}$ dove $b^{(t-1)}$ assume il valore definito dalla **3.26** con $g^{(t)} = -\nabla E^{(t)}$.

4. Se t=0 allora $l^{(t)} = .001$, altrimenti $l^{(t)}$ assume il valore definito dalla 3.35

5. Si calcoli $d^{(t)} = d^{(t)\mathrm{T}} H^{(t)} d^{(t)} + l^{(t)} \|d^{(t)}\|^2$; se $d^{(t)} \le 0$ allora $d^{(t)} = \bar{d}^{(t)}$ dove $\bar{d}^{(t)}$ assume il valore definito dalla **3.31**.

6. Si calcoli $\Delta^{(t)}$ con la **3.34** dove $a^{(t)} = d^{(t)\mathrm{T}} \nabla E^{(t)} / d^{(t)}$ e $g^{(t)} = -\nabla E^{(t)}$.

7. Se $\Delta^{(t)} > 0$ allora $w^{(t+1)} = w^{(t)} - a^{(t)} d^{(t)}$ altrimenti $w^{(t+1)} = w^{(t)}$.

8. Se $E(w^{(t+1)}) > e$ allora $t = t + 1$ e si ritorni al passo **2** altrimenti **STOP**.

Fig. 3.11 - algoritmo Scaled Conjugate Gradients.

3.6 Il metodo di Newton.

Se si utilizza un'approssimazione quadratica locale della funzione errore, si può ottenere un'espressione della posizione del minimo. Il gradiente in ogni punto *w* è infatti dato da:

$$g = \nabla E = H(w - w^*) \tag{3.36}$$

dove *w** corrisponde al minimo della funzione errore che soddisfa:

$$w^* = w - H^{-1} g \ . \tag{3.37}$$

Il vettore $-H^{-1}g$ è conosciuto come direzione di Newton ed è la base per una varietà di strategie di ottimizzazione tra cui il ***Quasi Newton Algorithm*** (QNA) che, invece di calcolare la matrice *H* e quindi la sua inversa, ne calcola un'approssimazione in una serie di passi successivi. Questo approccio genera una sequenza di matrici *G* che sono approssimazioni sempre più accurate di H^{-1}, usando solo informazioni relative alla derivata prima della funzione errore. Il problema di avere una matrice *H* non positiva è risolto partendo da una matrice unitaria (positiva per definizione) ed assicurandosi che le approssimazioni successive mantengano positiva questa matrice.

Dalla *formula di Newton* (3.37) notiamo che i vettori peso ai passi *t* e *t*+1 sono correlati ai corrispondenti gradienti dalla formula:

$$w^{(t+1)} - w^{(t)} = -H^{-1}(g^{(t+1)} - g^{(t)}) \tag{3.38}$$

che è conosciuta come *Quasi Newton Condition*. L'approssimazione *G* è quindi costruita così da soddisfare questa condizione. La formula per *G* è quindi:

$$G^{(t+1)} = G^{(t)} + \frac{pp^{\mathrm{T}}}{p^{\mathrm{T}}v} - \frac{(G^{(t)}v)v^{\mathrm{T}}G^{(t)}}{v^{\mathrm{T}}G^{(t)}v} + (v^{\mathrm{T}}G^{(t)}v)uu^{\mathrm{T}} \tag{3.39}$$

dove sono stati utilizzati i seguenti vettori:

$$p = w^{(t+1)} - w^{(t)}, \quad v = g^{(t+1)} - g^{(t)} \quad \text{e} \quad u = \frac{p}{p^{\mathrm{T}}v} - \frac{G^{(t)}v}{v^{\mathrm{T}}G^{(t)}v}. \tag{3.40}$$

Inizializzare la procedura utilizzando la matrice identità corrisponde a considerare, al primo passo, la direzione del gradiente negativo mentre, ad ogni passo successivo, la direzione $-Gg$ è garantita essere una direzione discendente. L'espressione 3.38 potrebbe però portare la ricerca al di fuori dell'intervallo di validità dell'approssimazione quadratica. La soluzione è di utilizzare il *line search* per cercare il minimo della funzione lungo la direzione di ricerca. Utilizzando questo sistema, i pesi verrebbero aggiornati in questo modo:

$$w^{(t+1)} = w^{(t)} + a^{(t)}G^{(t)}g^{(t)} \tag{3.41}$$

dove α è ottenuto dal *line search*. Si veda la figura 3.12 per una descrizione formale dell'algoritmo di apprendimento QNA.

Un vantaggio del QNA rispetto ai *gradienti coniugati* è che il *line search* non deve calcolare necessariamente a con grande precisione dato che questo non è un fattore critico per l'algoritmo. Uno svantaggio è che richiede l'utilizzo di una matrice G $|w|\times|w|$ che porta ad uno spreco enorme di memoria per $|w|$ molto grande. Un modo per ridurre lo spazio di memoria occupato da G è quello di sostituire ad ogni passo G con una matrice unitaria. Effettuando questa sostituzione in 3.39 e moltiplicando per g (il gradiente corrente), otteniamo:

$$d^{(t+1)} = -g^{(t)} + Ap + Bv \tag{3.42}$$

Dove A e B sono valori scalari definiti come:

$$A = -\left(1 + \frac{v^T v}{p^T v}\right)\frac{p^T g^{(t+1)}}{p^T v} + \frac{v^T g^{(t+1)}}{p^T v} \quad \text{e} \quad B = \frac{p^T g^{(t+1)}}{p^T v} \tag{3.43}$$

Si consideri una **MLP** e si indichi con $w^{(t)}$ il vettore di pesi al tempo t.

1. Si inizializzi $w^{(0)}$ con valori piccoli e casuali, si setti e e si pongano t=0 e $G^{(0)} = I$.

2. Si presenti tutto il *training set* e si calcoli $E(w^{(t)})$ ovvero la funzione di errore rispetto alla configurazione attuale dei pesi.

3. Se t=0 allora $d^{(t)} = -\nabla E^{(t)}$ altrimenti $d^{(t)} = -G^{(t-1)}\nabla E^{(t-1)}$.

4. Si calcoli $w^{(t+1)} = w^{(t)} - a d^{(t)}$ dove a è ottenuto dal line search.

5. Si calcoli $G^{(t+1)}$ tramite l'equazione **3.39**.

6. Se $E(w^{(t+1)}) > e$ allora $t = t+1$, si calcoli e si ritorni al passo **2** altrimenti **STOP**.

Fig. 3.12 - Quasi-Newton Method.

Si consideri una **MLP** e si indichi con $w^{(t)}$ il vettore di pesi al tempo *t*.

1. Si inizializzi $w^{(0)}$ con valori piccoli e casuali, si setti la costante *e* e si pongano *t*=0.

2. Si presenti tutto il *training set* e si calcoli $E(w^{(t)})$ ovvero la funzione di errore rispetto alla configurazione attuale dei pesi.

3. Se *t*=0 allora $d^{(t)} = -\nabla E^{(t)}$ altrimenti $d^{(t)} = -\nabla E^{(t-1)} + Ap + Bv$.

4. Si calcoli $w^{(t+1)} = w^{(t)} - \alpha d^{(t)}$ dove α è ottenuto dal line search.

5. Si calcolino *A* e *B* per la prossima iterazione come descritto nella **3.43**.

6. Se $E(w^{(t+1)}) > e$ allora $t = t + 1$, si calcoli e si ritorni al passo **2** altrimenti **STOP**.

Fig. 3.13 - Quasi-Newton Method a memoria limitata.

mentre *p* e *v* sono i vettori definiti nella 3.40.

È da notare che se il *line search* ritorna valori esatti allora la 3.42 produce direzioni di ricerca mutuamente coniugate. Per valori di α approssimati, la precedente proprietà non vale anche se l'algoritmo si comporta comunque bene. La figura 3.13 mostra l'algoritmo risultante da questo nuovo approccio.

Riferimenti.

R. Beale, T. Jackson "Neural Computing: An Introduction" *Institute of Physics Publishing Bristol and Philadelphia*, 1980.

C. M. Bishop "Neural Networks for Pattren Recognition" *Oxford University Press*, 1995.

D. Rumelhart, G Hinton, R.Williams "Learning Internal Representations by Error Propagation" in *ICS Report 8506 Institute of Cognitive Science University of California*, 1985.

CAPITOLO 4

Apprendimento Non Supervisionato.

4.1 Introduzione.

In questo capitolo ci si propone di illustrare diversi modelli di rete neurale tutti caratterizzati da un apprendimento *non supervisionato.* Questo tipo di apprendimento (come già spiegato nel paragrafo 2.4) conduce alla creazione di una *tassonomia*, vale a dire di una classificazione degli esempi in categorie di somiglianza. Ovviamente gli esempi forniti alla rete durante l'apprendimento non sono organizzati in coppie input/output, ma sono una serie di soli input (non esiste un insieme di etichette). La rete sviluppa in modo autonomo una organizzazione tale che input simili fanno attivare neuroni tra loro vicini, mantenendo rilassati i neuroni lontani.

Il primo algoritmo mostrato è quello di *Kohonen* che presenta sorprendenti analogie con alcuni comportamenti del cervello. Si discuterà poi, come si può modificare questo algoritmo per ottenere prestazioni nettamente superiori. Si cercherà, inoltre, di risolvere in modo automatico un problema annoso e non trascurabile delle reti non supervisionate ovvero il *labeling* delle classi ottenute in fase di apprendimento.

4.2 Mappe caratteristiche del cervello e reti di Kohonen.

La rappresentazione dei dati nel cervello umano è in generale quasi del tutto sconosciuta. Gli studi fatti in tale campo hanno comunque evidenziato come nel cervello vi siano zone distinte, ognuna dedicata ad un particolare tipo di processing, in particolare esiste una zona dedicata alla visione, una al moto degli arti, ecc.

Un'analisi della zona del cervello dedicata all'udito ha mostrato che la corteccia cerebrale di questa zona presenta una risposta alle eccitazioni sonore molto particolare infatti, le aree di corteccia che presentano un'attività maggiore in corrispondenza di uno stimolo sonoro ad una determinata frequenza, sono distribuite in maniera tale che aree vicine sono sensibili a frequenze vicine.

La rete di neuroni presente nella corteccia cerebrale è essenzialmente costituita da uno strato bidimensionale di unità di calcolo fittamente interconnesse tra di loro. Si stima che ogni neurone abbia circa 10000 interconnessioni e che sviluppi una particolare attitudine a rispondere a certi stimoli mediante un particolare tipo di interazione con i neuroni vicini, tale interazione presenta un andamento del tipo mostrato in figura 4.1. Per essere più chiari diremo che le interazioni presentano essenzialmente tre tipi di comportamento: fortemente *eccitatorio* lungo un raggio di influenza abbastanza ridotto (50-100 micron); debolmente *inibitorio* in un'area che circonda la zona di interazione di tipo eccitatorio ma di estensione maggiore (200-500 micron); ancora *eccitatorio* ma in maniera molto debole con neuroni lontani anche diversi centimetri.

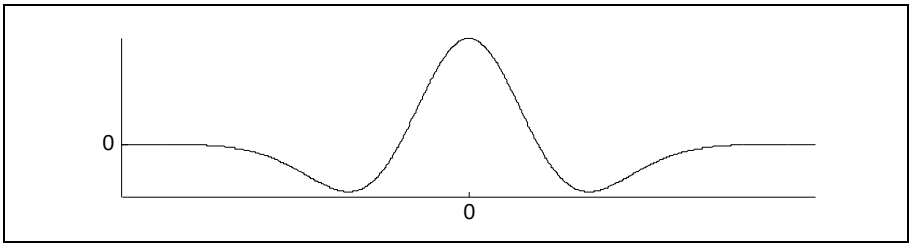

Fig. 4.1 - funzione a cappello messicano.

La struttura e il funzionamento del modello biologico sopra descritto furono ricalcati da **Kohonen** (professore di Informatica all'università di Helsinki) nella formulazione dei principi di un particolare tipo di rete neurale che da lui prende il nome (la stessa rete si indica anche col termine **SOM** da *Self-Organizing Map*). La figura 4.2 mostra una tipica rete di Kohonen: come si può vedere si tratta di una rete di due strati totalmente connessi dove il secondo strato non è lineare ma è organizzato su una griglia piatta (ciò permette una memorizzazione capace di mantenere una sorta di relazione topologica tra i dati). Ogni neurone dello strato di output (il secondo) comunica anche con i suoi vicini tramite le cosiddette *connessioni laterali* che sono di

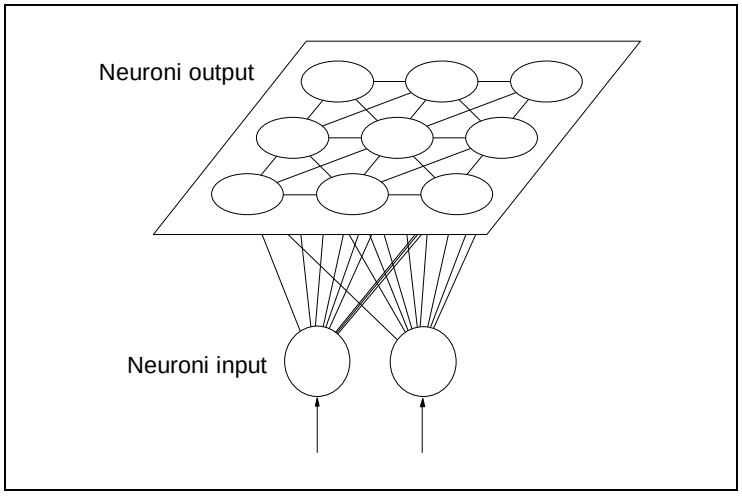

Fig. 4.2 - struttura di una rete di Kohonen.

tipo inibitorio e il cui scopo è quello di far sì che le unità entrino in competizione tra loro in modo che solo quelle maggiormente attivate riescano ad emettere un segnale in uscita mentre le altre avranno uscita nulla. Per questo motivo si parla anche di *apprendimento competitivo*.

Quando un pattern *x* viene presentato alla rete, ogni neurone output *i* ne riceve le componenti e ne calcola la distanza dal suo vettore di pesi w_i. Analogamente al comportamento delle reti biologiche, l'unità che presenta la minima distanza dal pattern in input sarà quella che avrà una risposta maggiore e, inibendo i neuroni adiacenti, sarà l'unica a sparare. In fase di apprendimento, la rete analizza ripetutamente i pattern di training e, per ognuno di essi, viene prima trovato il neurone vincente *k*, poi vengono modificati i suoi pesi e quelli dei neuroni vicini in modo da avvicinarli ulteriormente all'input appena processato. In altre parole la rete rafforza la sua convinzione che l'input *x* debba appartenere alla classe rappresentata dai neuroni in un intorno del *k*-esimo (che ha sparato). La rete si organizza quindi in modo che pattern simili fanno vincere neuroni vicini. L'algoritmo di training è descritto più formalmente in figura 4.3.

Come si può notare, non esiste un parametro che misura il grado di convergenza dell'algoritmo e, in generale, il processo di apprendimento viene fatto terminare dopo

Si consideri una **SOM** costituita da una griglia di *m* neuroni con *n* input. Sia $x_i^{(t)}$ il valore dell'*i*-esimo neurone di input al tempo *t* e $w_{ij}^{(t)}$ il peso tra l'*i*-esimo neurone input e il *j*-esimo neurone output sempre al tempo *t*.

1. Si inizializzino tutti i pesi $w_{ij}^{(0)}$ con valori piccoli e casuali.

2. Si presenti alla rete un nuovo pattern $x^{(t)} = \left(x_1^{(t)}, x_2^{(t)}, \ldots, x_n^{(t)}\right)$ settando opportunamen-te i valori dei neuroni di input.

3. Si calcoli la distanza euclidea tra il vettore in input e il vettore dei pesi per ogni nodo *j* con la formula:

$$d_j = \sum_{i=1}^{n} \left(x_i^{(t)} - w_{ij}^{(t)}\right)^2$$

4. Si selezioni il nodo *k* con la minima distanza d_k.

5. Si aggiornino tutti i pesi con la seguente formula: $w_{ij}^{(t+1)} = w_{ij}^{(t)} + \Delta w_{ij}^{(t)}$ dove

$$\Delta w_{ij}^{(t)} = e^{(t)} \cdot h_{s^{(t)}}\left(d(j,k)\right) \cdot \left(x_i^{(t)} - w_{ij}^{(t)}\right)$$

Il termine $e^{(t)}$ è il *gain term* $\left(0 \le e^{(t)} < 1\right)$ che decresce nel tempo per rallentare l'adattamento dei pesi; la funzione $h_{s^{(t)}}(x)$ è una funzione unimodale con varianza $s^{(t)}$ che decresce al crescere di *x*; $d(j,k)$ indica la distanza sulla griglia tra il neurone *k* e il neurone *j*.

6. Si ripeta il procedimento tornando al passo **2**.

Fig. 4.3 - algoritmo di apprendimento di una rete di Kohonen.

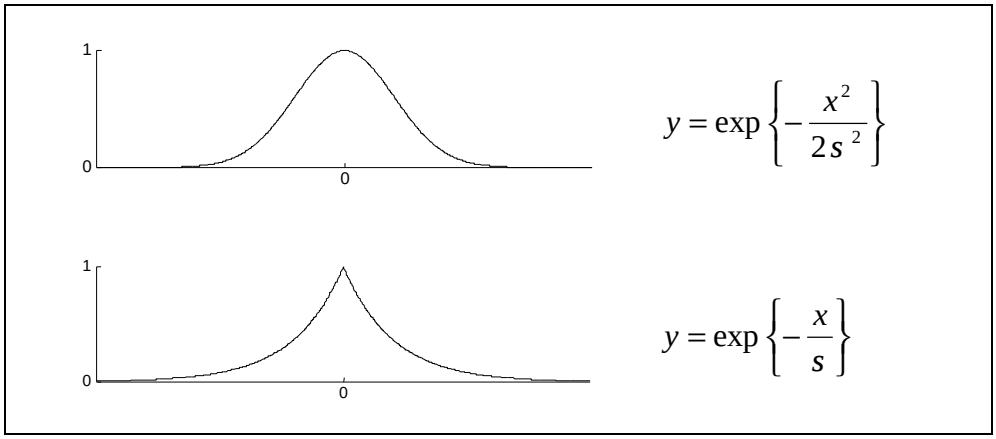

Fig. 4.4 - esempi di funzioni di adattamento.

un numero finito di iterazioni stabilito a priori. Per quanto riguarda la funzione $h_{s(t)}(x)$ utilizzata al punto 5, essa può essere sia la funzione a cappello messicano di figura 4.1, sia una funzione più semplice da calcolare che ne approssima l'andamento (vedi figura 4.4). Questo tipo di funzione permetterà che i pesi si adattino all'input in maniera decrescente via via che ci si allontana dal neurone che ha sparato. La rete cerca infatti di formare sulla griglia una *regione* che risponda ad uno spazio di valori intorno quel particolare input, la qual cosa comporta che, al termine dell'apprendimento, la griglia di output sarà organizzata in regioni, ognuna delle quali sarà ricettiva a pattern simili. Si renderà necessaria, ad una fase successiva, l'etichettamento manuale dei neuroni della griglia con il nome di una classe (operazione di *labeling*) in modo da identificare il sottoinsieme rappresentato da ogni regione.

La rete di Kohonen, così com'è, è stata utilizzata con successo nel campo dell'analisi di segnali vocali e nella classificazione di immagini. Nonostante ciò si possono apportare ad essa delle migliorie per ottenere prestazioni maggiori e soprattutto per evitare, almeno in parte, il *labeling* manuale. Vedremo come nei prossimi paragrafi.

4.3 Algoritmo di apprendimento "Neural-Gas".

L'algoritmo di apprendimento *Neural-Gas* è un sostanziale miglioramento dell'algoritmo di Kohonen infatti non solo converge più rapidamente ma raggiunge anche un valore di *distorsione media*[1] più basso. Esso fu presentato nel 1993 da Thomas M. Martinetz e utilizza un tipo di adattamento *soft-max* dei pesi (si aggiornano anche i pesi di neuroni non vincenti) ma, invece di variare la percentuale di adattamento a seconda della distanza sulla griglia dal neurone che ha sparato, si genera una sorta di classifica dei neuroni in base alla vicinanza all'input e si aggiornano i pesi a seconda della posizione di ogni neurone in questa classifica.

[1] Sia $P(x)$ la distribuzione di probabilità dei pattern sull'insieme $V \subseteq \Re^n$ e sia $w_{i(x)}$ il vettore di pesi del neurone che classifica il pattern x, si definisce *distorsione media* la quantità:

$$E = \int P(x)\left(x - w_{i(x)}\right)^2 d^n x$$

Si consideri l'algoritmo in **figura 4.3** e si sostituiscano i passi **4** e **5** con i seguenti:

4. Si compili una classifica $(d_{j_1}, d_{j_2}, \ldots, d_{j_m})$ delle distanze ottenute al passo **3** tale che $d_{j_1} \le d_{j_2} \le \ldots \le d_{j_m}$ e si indichi con $rank(j)$ la posizione che occupa d_j in questa classifica.

5. Si aggiornino tutti i pesi con la seguente formula: $w_{ij}^{(t+1)} = w_{ij}^{(t)} + \Delta w_{ij}^{(t)}$ dove

$$\Delta w_{ij}^{(t)} = e^{(t)} \cdot h_{s^{(t)}}(rank(j)) \cdot \left(x_i^{(t)} - w_{ij}^{(t)}\right)$$

Tutti i termini hanno lo stesso significato che avevano nel punto **5** di **figura 4.3**.

Fig. 4.5 - algoritmo di apprendimento Neural-Gas.

L'algoritmo è formalmente descritto in figura 4.5. È da notare che non è necessario che i neuroni dello strato di output siano organizzati su una griglia bidimensionale dato che l'algoritmo non mantiene alcuna relazione topologica. Si considera quindi una normale rete su due strati lineari. Un problema che sopraggiunge è quello del *labeling* che diventa ancora più arduo dato che i neuroni che rappresentano la stessa classe non sono più concentrati nella stessa zona spazialmente identificabile ma sono sparsi sullo strato di output e sono quindi più difficilmente raggruppabili.

Risolveremo questo problema nel paragrafo 4.6; per il momento è ancora necessario sapere che il nome Neural-Gas deriva dal fatto che la dinamica di una rete di questo tipo ricorda la dinamica di un insieme di particelle diffuse in un potenziale dove le particelle occupano le posizioni rappresentate dai vettori peso dei neuroni della rete mentre il potenziale è ottenuto dando ad ogni vettore di training potenziale negativo (quindi attrattivo nei confronti delle particelle) e ad ogni particella potenziale positivo (le particelle si respingono tra loro).

4.4 Altri algoritmi di apprendimento non supervisionato.

Accenneremo brevemente in questo paragrafo ad altri due approcci all'apprendimento non supervisionato utilizzati nel presente lavoro di tesi. Essi hanno la comune caratteristica (a differenza dell'algoritmo di Kohonen) di applicare la tecnica del *gradiente discendente* su una funzione d'errore.

Il primo approccio è l'algoritmo di clustering ***K-means*** nella sua versione *on-line* che applica il gradiente discendente alla funzione *distorsione media*:

$$E = \int P(x)\left(x - w_{i(x)}\right)^2 d^n x$$

dove ogni simbolo assume il significato descritto nella nota 1. Il corrispondente algoritmo di apprendimento è dato in figura 4.6. Il principale limite di questa tecnica è che la funzione *E*, tranne che in casi veramente banali, presenta minimi locali che possono arrestare il processo di apprendimento prima che la rete raggiunga la

Si consideri l'algoritmo in **figura 4.3** e si sostituisca il solo passo **5** con il seguente:

5. Si aggiornino i pesi del *k*-esimo neurone come segue: $w_{ik}^{(t+1)} = w_{ik}^{(t)} + \Delta w_{ik}^{(t)}$ dove

$$\Delta w_{ik}^{(t)} = e^{(t)} \cdot \left(x_i^{(t)} - w_{ik}^{(t)} \right)$$

Il termine $e^{(t)}$ ha lo stesso significato che aveva nel punto **5** di **figura 4.3**.

Fig. 4.6 - algoritmo di apprendimento K-means.

Nell'algoritmo in **figura 4.3**, si elimini il passo **4** e si sostituisca il passo **5** con il seguente:

5. Si aggiornino tutti i pesi con la seguente formula: $w_{ik}^{(t+1)} = w_{ik}^{(t)} + \Delta w_{ik}^{(t)}$ dove

$$\Delta w_{ij}^{(t)} = e^{(t)} \cdot \frac{\exp\left(-b^{(t)} d_j\right)}{\sum_{k=1}^{m} \exp\left(-b^{(t)} d_k\right)} \cdot \left(x_i^{(t)} - w_{ij}^{(t)} \right)$$

Il termine $e^{(t)}$ ha lo stesso significato che aveva nel punto **5** di **figura 4.3**; il termine $b^{(t)}$ ha inizialmente valore molto elevato per evitare i minimi locali ma deve essere progressivamente diminuito fino a raggiungere valore nullo.

Fig. 4.7 - algoritmo di apprendimento Maximum Entropy.

configurazione ottimale (essa si raggiunge solo quando i vettori peso dei neuroni assumono valori che rendono minima la funzione *E*).

Il secondo approccio è l'algoritmo ***Maximum Entropy*** che applica il *gradiente discendente* (con adattamento *soft-max* dei pesi per evitare i minimi locali) alla funzione d'errore:

$$E_{me} = -\frac{1}{b} \int P(x) \ln \sum_{j=1}^{m} e^{-b\left(x - w_j\right)^2} d^n x.$$

Il termine β è detto *temperatura inversa* ed il suo significato è spiegato in figura 4.7 dove si presenta anche il corrispondente algoritmo di apprendimento.

È da notare che anche l'algoritmo *Neural-Gas* applica il *gradiente discendente* con adattamento *soft-max* dei pesi. La funzione di errore che minimizza è:

$$E_{ng} = \frac{1}{2C(s)} \sum_{j=1}^{m} \int P(x) \cdot h_s\left(rank(x)\right) \cdot \left(x - w_j\right)^2 d^n x \quad \text{con} \quad \mathrm{C}(s) = \sum_{k=0}^{m-1} h_s(k).$$

4.5 Growing-Cell Structures.

Un limite delle reti neurali considerate fino a questo punto è una intrinseca rigidità strutturale. La morfologia della rete deve essere sempre stabilita a priori e non può variare nel tempo. Ciò comporta che il progettista debba perdere una notevole quantità

di tempo nel ricercare la dimensione ottimale per la rete in modo che questa non sia tanto piccola da essere incapace di generalizzare ma neanche tanto grande da rallentare il processo di apprendimento. Una rete ***GCS*** (*Growing-Cell Structures*) è una rete ad apprendimento non supervisionato che risolve (o almeno si propone di risolvere) questo problema. Essa è infatti in grado di alterare la propria struttura in modo da espandersi o restringersi a seconda delle necessità.

Scopo della rete è effettuare un mapping dallo spazio dei pattern $V \subseteq \Re^n$ sul quale è definita una distribuzione di probabilità $P(x)$ su una struttura bidimensionale discreta A in modo tale che:

- pattern simili siano rappresentati da elementi topologicamente vicini di A;
- elementi vicini di A rappresentino pattern simili;
- Le regioni di V con più alta densità di probabilità vengano rappresentate da un numero maggiore di elementi di A.

La struttura A è un *simplesso* bidimensionale (vedi figura 4.8) i cui vertici sono i neuroni e i cui archi ne denotano le relazioni di vicinanza topologica. Ogni modifica alla rete (aggiunta o rimozione di nodi) mantiene sempre intatte le proprietà del *simplesso* (la struttura risultante è ancora un *simplesso*). Ad ogni neurone è associato un vettore di pesi che ne rappresenta la posizione nello spazio V e che viene utilizzato in fase di *testing* per calcolare l'output della rete. Il *testing* consiste infatti nel presentare un pattern appartenente a V alla rete e nel far sparare il solo neurone più vicino. Niente di diverso da quanto accadeva nelle altre reti. Le differenze sono tutte nell'algoritmo di apprendimento che, partendo da un *simplesso* banale con tre nodi (vedi figura 4.8b), cerca di ottenere una rete ottimale attuando un processo di crescita controllata con rimozione occasionale di nodi.

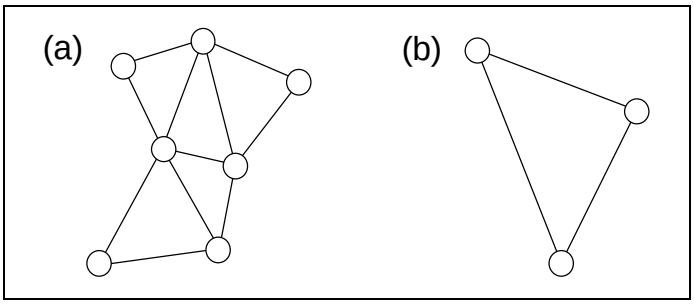

Fig. 4.8 - esempi di simplessi bidimensionali.

Il processo di adattamento è descritto formalmente (ad esclusione della fase di rimozione) in figura 4.9 ma può essere sintetizzato in questo modo: si determina il neurone vincente per l'input corrente e si avvicinano i pesi del vincente e dei suoi vicini all'input. La forza dell'adattamento è costante nel tempo. Più specificamente si utilizzano due parametri: ε_b per il *best-matching* e ε_n per i vicini.

Si consideri una rete **GCS** costituita da un simplesso con *m* nodi ad *n* input. Dato l'*i*-esimo neurone, si indichi con w_i il suo vettore di pesi, con τ_i un contatore ad esso associato che prende il nome di *signal counter* e con N_i l'insieme dei suoi vicini topologici. Si fissi inoltre un intero λ che indicherà il numero di step tra due inserimenti consecutivi di nodi.

1. Si inizializzi la rete con una struttura del tipo mostrato in figura 4.8b dove i vettori peso dei tre neuroni (*m*=3) hanno valori piccoli e casuali e i *signal counter* sono azzerati.

2. Si presenti in input alla rete un nuovo pattern $x \in V$ e si selezioni il nodo *k* tale che:

$$\|w_k - x\| \leq \|w_i - x\| \quad i = 1,2, \ \ldots \ , m.$$

3. Si aggiornino i pesi del neurone vincente e dei suoi vicini topologici come segue:

$$w_k = w_k + e_b \cdot (x - w_k); \quad w_i = w_i + e_n \cdot (x - w_i) \quad \forall i \in N_k$$

dove i termini ε_b e ε_n sono parametri di adattamento fissati a-priori.

4. Si incrementi il *signal counter* del neurone vincente e si decrementi quello di tutti gli altri neuroni in questo modo: $\tau_k = \tau_k + 1; \ \tau_i = \tau_i - \alpha\tau_i \quad i = 1,2, \ \ldots \ , m$; dove α è un parametro compreso tra 0 e 1 che determina il decadimento del *signal counter*.

5. Se il numero di step effettuati non raggiunge λ, si ripeta il procedimento tornando al passo **2** altrimenti si prosegua con il passo **6**.

6. Si calcoli per ogni neurone *i* della rete la *frequenza relativa* f_i e si individui il neurone *k* con *frequenza relativa* maggiore dove:

$$f_i = \tau_i \Big/ \sum_{j \in A} \tau_j \, .$$

7. Si cerchi in N_k il neurone *s* che abbia distanza maggiore da *k* e si inserisca un nuovo nodo *r* tra *s* e *k* (vedi **figura 4.10**) ponendo $w_r = (w_k + w_s)/2$. Si ripristino i collegamenti in questo modo: la connessione tra *s* e *k* viene rimossa mentre *r* viene connesso ad *s*, a *k* e a tutti i vicini comuni ad *s* e *k*.

8. Si ricalcolino i *signal counter* in questo modo:

$$\tau_i = \tau_i + \Delta\tau_i \quad \text{dove} \quad \Delta\tau_i = \frac{F_i^{new} - F_i^{old}}{F_i^{old}} \cdot \tau_i \quad \forall i \in N_r; \quad \tau_r = -\sum_{i \in N_r} \Delta\tau_i$$

Il termine F_i è il volume della regione di Voronoi associata all'*i*-esimo neurone e si approssima in questo modo:

$$F_i \cong \left(\frac{1}{card(N_i)} \cdot \sum_{j \in N_i} \|w_j - w_i\| \right)^n$$

9. Si ritorni al punto **2** dopo aver azzerato il contatore di step.

Fig. 4.9 - algoritmo di apprendimento di una rete GCS.

Se si scelgono valori piccoli per ε_b e ε_n, le celle muoveranno dalla loro posizione iniziale a una posizione che manterranno in equilibrio dinamico senza smettere di muoversi completamente dato che i parametri non vengono mai diminuiti.

L'inserimento di un nuovo nodo avviene dopo un numero fissato λ di step di adattamento. Il nuovo nodo viene inserito tra il neurone che recentemente ha sparato più di tutti gli altri e il più distante tra i suoi vicini topologici (vedi figura 4.10). Questa scelta ha lo scopo di alleggerire il lavoro di un neurone che classifica troppo affiancandogliene un altro.

Non è presente nell'algoritmo di apprendimento una condizione di terminazione, esso può infatti essere interrotto in qualsiasi istante o dopo un numero prefissato di iterazioni. La struttura che rappresenta la rete è in ogni istante consistente con la specifica di simplesso, questo comporta che si può anche riprendere l'apprendimento dopo una interruzione.

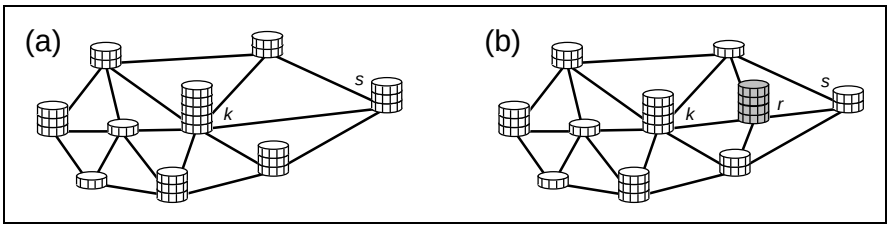

Fig. 4.10 - inserimento di un nodo in una GCS (le pile sono i valori di τ_i).

La *rimozione* di celle non è sempre necessaria, essa può risultare utile quando la distribuzione di probabilità *P(x)* consta di regioni separate a densità di probabilità positiva. In questo caso si cerca di rimuovere i soli neuroni superflui ovvero quelli che si trovano in zone con bassa densità di probabilità. A questo scopo si cerca di approssimare la densità di probabilità della zona rappresentata dal generico neurone *i* con la formula: $\tilde{p}_i = f_i / F_i$. L'algoritmo di figura 4.9 può essere quindi modificato aggiungendo una procedura che rimuove periodicamente i neuroni con valori di $\tilde{p}_i$ inferiori ad una qualche soglia η premurandosi di conservare sempre la consistenza della struttura pena il malfunzionamento dell'algoritmo di apprendimento per gli step successivi.

4.6 Reti gerarchiche e *labeling* automatico.

Abbiamo accennato nei paragrafi precedenti al *labeling*, operazione che si effettua manualmente al termine dell'algoritmo di apprendimento e che consiste nell'associare ad ogni neurone della rete un'etichetta di classe. Se la rete avesse un numero di neuroni esattamente uguale al numero di classi che si attendono in output, il *labeling* si ridurrebbe al dare in input alla rete un pattern per classe e ad etichettare ogni volta il neurone che si attiva usando il nome stesso della classe. Sfortunatamente, il numero di neuroni che è necessario utilizzare in una rete non supervisionata (almeno per le tipologie considerate in questo capitolo) deve essere superiore di almeno tre o quattro volte il numero di classi previste, altrimenti si avrebbero gravi ripercussioni sul processo di apprendimento. In fase di training infatti, ogni qual volta un neurone vince, la rete aggiorna non solo i pesi del vincitore ma anche quelli di altri neuroni

(adattamento soft-max) per evitare lo stallo in minimi locali della funzione d'errore (vedi paragrafo 4.4). Con un numero troppo basso di neuroni, alla presentazione di un nuovo pattern si andranno a modificare anche i pesi di neuroni rappresentanti di altre classi. Ciò comporterà che classi più forti (con più pattern nel training set) saranno rappresentate da più neuroni e che classi più deboli non avranno alcun rappresentante. Si parla in questo caso di *rete sottodimensionata* e di conseguenti *insufficienti capacità di generalizzazione*.

È quindi necessario utilizzare un gran numero di neuroni rispetto al numero di classi e, come conseguenza, il *labeling* si complica dato che la stessa classe ha più rappresentanti sparsi sullo strato di output. Per identificali tutti, dovremmo provare a dare in pasto alla rete più di un pattern per ogni classe con il rischio di far attivare sempre lo stesso neurone. In questo caso potranno rimanere neuroni di cui non riusciamo a definire l'appartenenza. Tutto ciò può essere evitato utilizzando le reti gerarchiche non supervisionate.

Una ***ML-SOM*** (*MultiLayer Self-Organizing Map*) è una rete neurale organizzata in più strati contenenti ognuno una *SOM* (vedi paragrafo 4.2). Il numero di nodi in ogni strato decresce via via che ci si avvicina al livello output. La rete ha per questo motivo forma piramidale (vedi figura 4.11). I neuroni input hanno il solo compito di ricevere l'input dall'esterno e di propagarlo al primo strato competitivo. L'apprendimento del primo strato può avvenire quindi come in una normale rete di Kohonen. Una volta terminato, i pesi risultanti vengono propagati come input allo strato successivo che su di essi baserà il proprio apprendimento. Lo stesso procedimento è quindi ripetuto fino a raggiungere lo strato più alto ovvero quello di output (per maggior chiarezza si veda la figura 4.12). Dopo di ciò, la rete può essere utilizzata.

Il normale funzionamento della rete consiste nel prendere un input dall'esterno e presentarlo al primo strato e, per ogni strato, trovare il neurone vincente e presentare

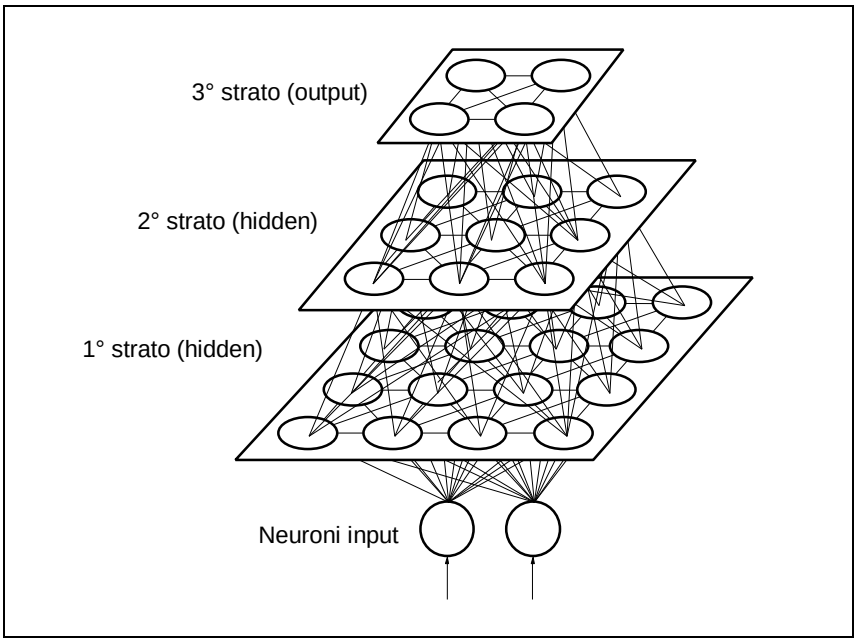

Fig. 4.11 - struttura di una rete ML-SOM.

come input del successivo strato il vettore di pesi di questo neurone. L'ultimo strato avrà il normale output di una *SOM* (1 per il neurone vincente, 0 per gli altri).

La *ML-SOM* è solo un esempio di rete gerarchica che si basa sull'algoritmo di apprendimento di una *SOM*. Utilizzando gli altri algoritmi visti in questo capitolo possiamo ottenere altrettante reti gerarchiche quali la *ML-MaximumEntropy*, la *ML-Kmean*s o ancora la *ML-NeuralGas* (o *Neural Gas Gerarchico*). Si può ancora pensare a reti gerarchiche dove ogni strato ha un diverso algoritmo di apprendimento in modo da sfruttare congiuntamente le caratteristiche positive di più algoritmi. Le possibilità combinatorie sono quindi pressoché infinite.

Non abbiamo ancora spiegato come queste reti possano aiutarci nel *labeling*. È semplice: basta creare una rete gerarchica che abbia nell'ultimo strato (quello di output) un numero di neuroni esattamente uguale al numero di classi. Il *labeling* in questo caso è del tutto banale. La *capacità di generalizzazione* della rete non è messa in pericolo da questa scelta dato che avremo dotato la rete di un primo strato sufficientemente grande e di strati successivi al primo grandi il necessario per ben classificare il ristretto insieme di pesi dello strato precedente. È ovvio quindi che il numero di neuroni in strati successivi non può decrescere bruscamente anche se questo significa avere più strati e, di conseguenza, un training più lento.

È da notare che non abbiamo accennato a reti *ML-GCS*. Come abbiamo visto, infatti, una rete gerarchica è utile quando l'ultimo strato ha un numero di neuroni esattamente uguale al numero di classi previste. Le reti *GCS*, per loro natura, hanno un numero variabile di neuroni e quindi non possono garantire questa condizione. Possiamo però sfruttare l'estrema velocità di apprendimento di questo tipo di rete progettando reti gerarchiche che abbiano uno o più strati *GCS* seguiti nella gerarchia da reti di tipo diverso purché lo strato di output sia *non-GCS* e abbia quindi una dimensione fissabile a priori.

Si consideri una **ML-SOM** con s strati e con training set X. Chiameremo X_i il training set dell'i-esimo strato e W_i l'insieme dei vettori pesi dei neuroni dell'i-esimo strato dopo il processo di training.

1. Si effettui il training del primo strato sull'insieme X (di conseguenza $X_1 = X$) utilizzando l'algoritmo di **figura 4.3**.

2. Utilizzando lo stesso algoritmo, si effettui il training dell'i-esimo strato ($i = 2, 3, \ldots, s$) sull'insieme $X_i \subseteq W_{i-1}$ che comprende i soli vettori di W_{i-1} corrispondenti a neuroni vincenti. Si considerano i soli neuroni vincenti perché sono gli unici ad essere rappresentati-vi, gli altri servono solo ad interpolare i valori dei neuroni vincenti.

Fig. 4.12 - algoritmo di apprendimento di una ML-SOM.

4.7 Algoritmi di apprendimento ibridi.

Le reti neurali non supervisionate, come è stato già detto nei precedenti paragrafi, sono in grado di effettuare il *clustering* di un insieme di *pattern*, ovvero di suddividere questo insieme in gruppi tali che il grado di *naturale associazione* sia alto tra i membri dello stesso gruppo e basso tra i membri di gruppi diversi. Queste caratteristiche permettono di annoverare questo tipo di reti neurali nella grande famiglia dei cosiddetti *algoritmi di clustering* che ha molti altri componenti e della quale le reti neurali rappresentano certamente l'esponente di spicco.

Gli *algoritmi classici* di clustering lavorano su tutto l'insieme di *pattern* possibili invece che su un *training set* rappresentativo dato che, a differenza delle reti neurali, non hanno capacità di generalizzazione. Ciò comporta un notevole spreco di tempo oltre che una classificazione molto rigida e non estensibile a nuovi pattern (se si presentano pattern diversi, si deve rifare il clustering). Di contro questi algoritmi possono vantare una notevole efficienza su insiemi piccoli ed è proprio da questo punto di vista che potrebbero tornarci utili. La figura 4.13 mostra un algoritmo generico di clustering di tipo agglomerativo che costituisce lo scheletro comune a molti metodi presenti in letteratura. Ciascun metodo si differenzia dagli altri, nel solo passo 2, per il modo in cui si calcolano le distanze tra due cluster diversi. La figura 4.14 mostra, a questo proposito, alcune possibili definizioni della funzione di distanza tra cluster.

Vediamo ora come utilizzare questi metodi per i nostri scopi. Non è nostra intenzione applicarli direttamente ai pattern in input dato che le reti neurali lavorano meglio e più velocemente. L'idea è quella di sfruttare questi metodi nel processo di *labeling* per generare una partizione automatica dell'insieme di neuroni dello strato di output in base alla vicinanza dei vettori peso. L'insieme da partizionare è quindi veramente limitato e questo tipo di tecnica ci garantisce risultati veloci e affidabili, paragonabili a quelli delle reti gerarchiche ma con tempi più brevi e con meno parametri da settare. Si potrebbe obbiettare che questa non è una tecnica neurale in senso stretto dato che non è possibile implementarla senza un controllo centrale che

Sia $W = \{w_1, w_2, \ldots, w_m\}$ un insieme di pattern da dividere in l cluster $C_1, \ldots, C_l$ con $l<m$, si proceda nel seguente modo:

1. Si conisderi una suddivisione di W in m clusters $C_1, \ldots, C_m$ tale che $C_i = \{w_i\}$.

2. Si calcoli la matrice delle distanza D tale che $D_{ij} = d(C_i, C_j)$.

3. Si individui l'elemento D_{ij} più piccolo della matrice D (se questo elemento non è unico se ne sceglie uno arbitrariamente) e si raggruppino i due cluster C_i e C_j in un nuovo cluster:

$$C_{ij} = C_i \cup C_j .$$

4. Se il numero di cluster ottenuti è maggiore di l si ritorni al **punto 2**; altrimenti **STOP**.

Fig. 4.13 - algoritmo generico di clustering di tipo agglomerativo.

Siano $C_i = \{w_{i1}, w_{i2}, \ldots, w_{in}\}$ e $C_j = \{w_{j1}, w_{j2}, \ldots, w_{jm}\}$ due cluster diversi, valutiamo la loro distanza:

a. metodo ***nearest neighbour***: $d(C_i, C_j) = \min_{1 \le k \le n,\ 1 \le l \le m} \|w_{ik} - w_{il}\|$;

b. metodo del ***centroide***: $d(C_i, C_j) = \left\| \frac{1}{|C_i|} \sum_{k=1}^{n} w_{ik} - \frac{1}{|C_j|} \sum_{k=1}^{n} w_{jk} \right\|$;

c. metodo della ***media tra gruppi***: $d(C_i, C_j) = \frac{1}{mn} \sum_{1 \le k \le n,\ 1 \le l \le m} \|w_{ik} - w_{jl}\|$.

Fig. 4.14 - alcune funzioni di distanza tra cluster.

vada a effettuare il clustering dei neuroni di output. Questo è vero ma, dato che non stiamo lavorando con reti neurali vere e proprie ma solo con *simulazioni software* di reti neurali, questo aspetto può essere trascurato rispetto al risparmio di tempo che, di contro, otteniamo. Nel paragrafo successivo, confrontando le prestazioni di queste reti con quelle delle reti gerarchiche, vedremo quanto è utile questo tipo di approccio.

4.8 Prestazioni delle reti non supervisionate.

In questo paragrafo cercheremo di mettere a confronto le prestazioni delle reti non supervisionate viste fino a questo momento provando ad effettuare un apprendimento su un insieme di pattern bidimensionali raggruppati in 6 *cluster* ben separati. Si tratta ovviamente di un caso non realistico dato che raramente i pattern sono così ben separabili anche ad occhio nudo. In applicazioni reali si ha generalmente a che fare con pattern di dimensioni molto grandi (anche > 100 nel caso specifico di questo lavoro di tesi) e con clusters soggetti ad *overlapping* (sovrapposizione). Il test che segue non è stato quindi effettuato allo scopo di preferire una rete rispetto all'altra già in questa fase ma ha il semplice obiettivo di analizzare più a fondo i comportamenti dei vari algoritmi finora mostrati.

Abbiamo considerato un *training set* composto da 230 *pattern* bidimensionali (le cui caratteristiche hanno valori compresi tra 0 e 400) distribuiti omogeneamente su 6 *cluster* ben separati (vedi figura 4.15a). Il *training set* è stato presentato 50 volte per ogni rete neurale per un totale di 1150 *adaption steps*. Ad ogni rete sono stati concessi 50 neuroni di output tranne che alla *SOM*. Essa ne ha avuti solo 49 distribuiti su una griglia 7x7 (non si poteva fare altrimenti).

Per i primi due algoritmi (*Kohonen* e *Neural-Gas*) è stata scelta come funzione di adattamento h_σ la *gaussiana* di figura 4.4a dove il termine σ è stato fatto decrescere esponenzialmente da 5 a .1 per la *SOM* e da 10 a .1 per la *Neural-Gas*. La differenza è dovuta al fatto che i neuroni della seconda rete sono organizzati linearmente su uno strato e quindi c'è bisogno di una maggiore forza di adattamento per smuovere neuroni

anche più distanti. Per quanto riguarda invece il termine β dell'algoritmo *Maximum Entropy*, esso è stato fatto variare sempre esponenzialmente da 5 a 100. Il *gain term* di tutte e tre queste reti più quello della rete *K-means* assume invece valori che decrementano da .7 a .05 in modo da passare da una prima fase di sgrossatura della rete nella quale i pesi possono subire variazioni anche consistenti per giungere via via ad una situazione in cui possono esserci solo variazioni minime. Questa seconda fase si chiama *fine tuning* e permette alla rete di perfezionare la rappresentazione interna del training set ottenuta nella fase precedente.

Un discorso a parte merita la rete *GCS*. Si è partiti in questo caso da un simplesso banale di 3 neuroni fino ad arrivare con 47 iterazioni del *training set* ad una rete di 50 neuroni. Il parametro λ è quindi stato settato esattamente pari alla grandezza del training set (230) per avere un inserimento alla fine di ogni *epoch*. I parametri di adattamento ε_b e ε_n sono stati fissati rispettivamente a .1 e a .01 mentre $\alpha = .05$.

Analizziamo ora il comportamento delle reti. La figura 4.15 mostra l'evoluzione della ***SOM*** dallo stato iniziale in cui i pesi dei neuroni hanno valori casuali (figura 4.15a) fino alla fine del processo di training. Come si può vedere, la rete cerca di assicurare una relazione topologica tra neuroni e dati muovendo inizialmente i pesi dei neuroni in modo da rispecchiare le posizioni degli stessi sulla griglia di output (figura 4.15b) per poi farli divergere fino a coprire gli spazi occupati dal training set (i rettangolini). Si può notare che non tutti i neuroni vengono utilizzati per rappresentare gli input, alcuni di essi rimangono infatti tra una classe e l'altra e andranno a classificare pattern ambigui, per i quali cioè non si riesce a dedurre la giusta classificazione.

La stessa cosa non avviene per la rete ***Neural-Gas*** infatti essa tende a sfruttare tutti i neuroni di cui è composta per rappresentare al meglio il *training set*. Come si può vedere della figura 4.16 e, più in dettaglio, dal grafico di figura 4.20, la rete raggiunge una buona rappresentazione dell'input già dopo poche iterazioni e tende a posizionare i suoi vettori peso sullo spazio delle caratteristiche in modo che tutti quelli che rappresentano la stessa classe siano tra loro il più lontano possibile (figura 4.16d). Ciò è facilmente spiegabile con la teoria (accennata già nel paragrafo 4.3) che assimila la dinamica di una rete Neural-Gas a quella di un insieme di particelle diffuse in un potenziale. Questo tipo di comportamento permette in genere di raggiungere valori più bassi della *distorsione media* rispetto alle altre reti.

Il comportamento dell'algoritmo ***K-means*** (figura 4.17) consiste essenzialmente nel presentare un pattern e avvicinare il neurone vincente, ciò implica che le posizioni di molti neuroni non verranno proprio modificate in fase di apprendimento. Alla fine avremo quindi uno scarso numero di neuroni rappresentativi con conseguente aumento della *distorsione media*. È un algoritmo molto veloce (ogni step dura circa un quarto rispetto agli algoritmi ad adattamento *soft-max*) ma molto inaffidabile.

La ***Maximum Entropy*** è una rete che tende a utilizzare in fase di training tutti i suoi neuroni (figura 4.18). Alla fine dell'apprendimento però i pesi dei neuroni sono tutti sovrapposti sui centroidi dei cluster (figura 4.18d), questo implica che solo pochi nodi (6 nel nostro caso) saranno ricettivi in fase di effettivo utilizzo della rete e dovranno

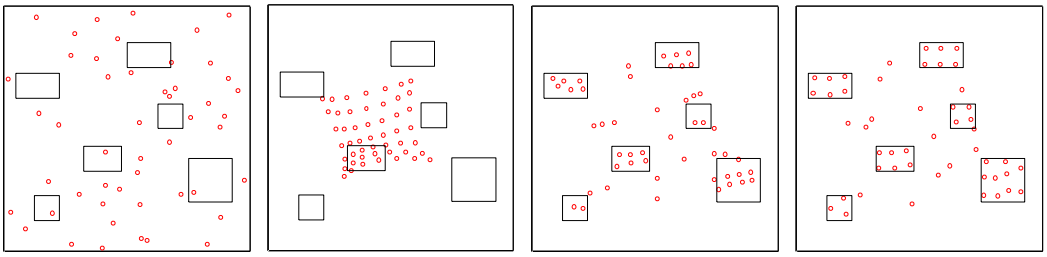

Fig. 4.15 - stato della rete SOM dopo 0, 10, 25 e 50 epoch.

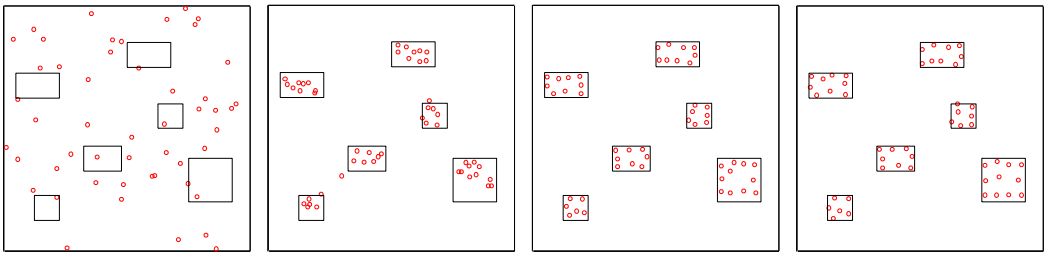

Fig. 4.16 - stato della rete Neural-Gas dopo 0, 10, 25 e 50 epoch.

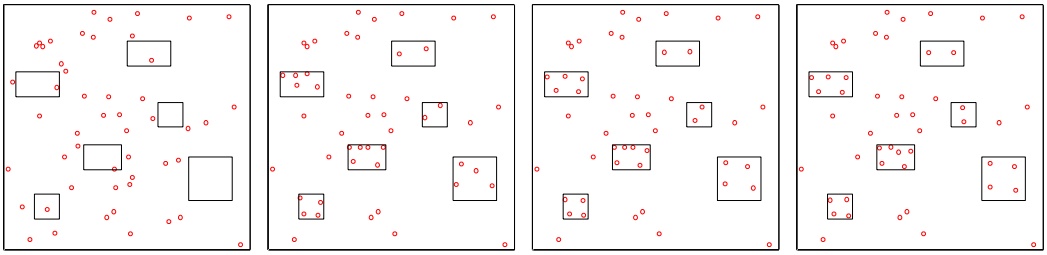

Fig. 4.17 - stato della rete K-means dopo 0, 10, 25 e 50 epoch.

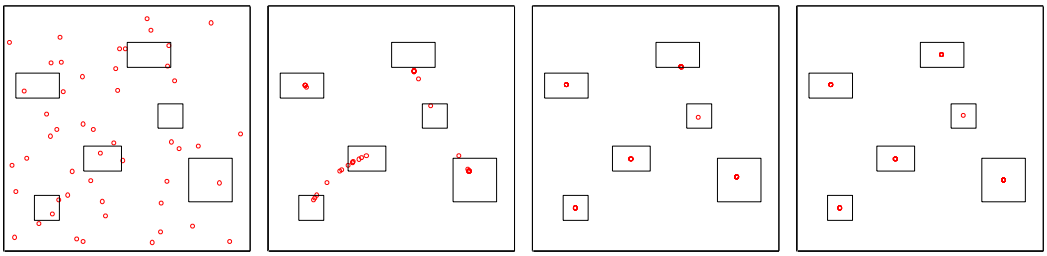

Fig. 4.18 - stato della rete Maximum Entropy dopo 0, 10, 25 e 50 epoch.

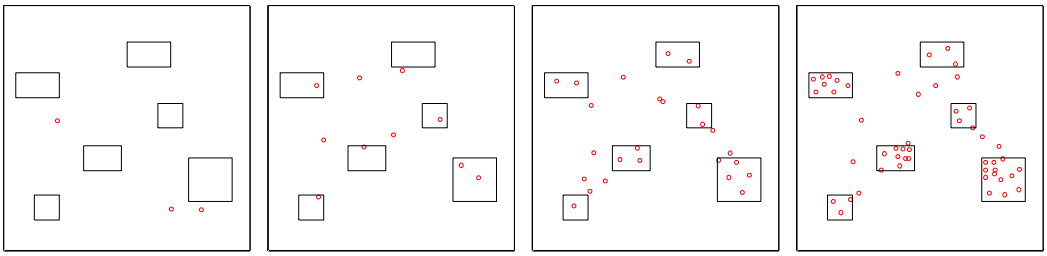

Fig. 4.19 - stato della rete Growing-Cell dopo 0, 10, 25 e 50 epoch.

rappresentare tutto lo spazio di input. La *distorsione media* sarà di conseguenza piuttosto elevato (vedi il grafico di figura 4.20).

Un pregio di questo tipo di rete potrebbe essere la facilità del *labeling* avendo un solo neurone per cluster. Un difetto è che la rete funziona bene solo per *cluster* ben separati.

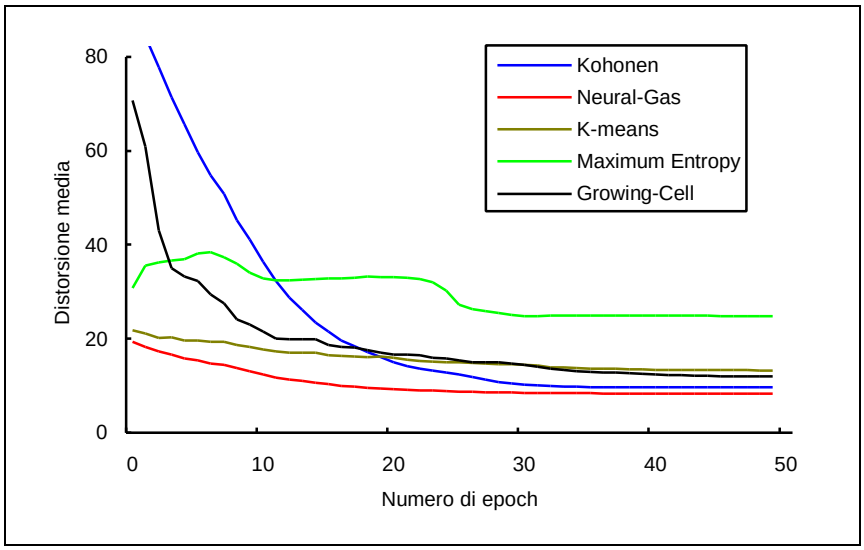

Fig. 4.20 - performances delle reti non supervisionate.

Non ci rimane che analizzare il comportamento della ***GCS*** (figura 4.19). Si tratta di un algoritmo estremamente veloce (più veloce del *K-means*) ma, nel contempo, sufficientemente affidabile. La velocità deriva in massima parte da due fattori. Primo: l'adattamento dei pesi non è di tipo *soft-max* in senso stretto; ad ogni iterazione non vengono aggiornati i pesi di tutti i neuroni ma solo quelli del neurone vincente e dei suoi immediati vicini con conseguente risparmio di tempo. Secondo: essendo la taglia della rete crescente, il calcolo delle distanze risulta velocissimo per le prime iterazioni perché ci sono pochi neuroni e solo nelle ultime si raggiungono velocità paragonabili a quelle degli altri algoritmi. Il valore della *distorsione media* raggiunto alla fine dell'apprendimento (vedi figura 4.20) è inoltre sufficientemente basso anche se comunque superiore sia a *Kohonen* che a *Neural-Gas*.

Vediamo ora come effettuare il ***labeling*** dei 50 neuroni ottenuti in fase di *training* con uno degli algoritmi precedenti, in modo da ottenere una effettiva ripartizione dello spazio delle caratteristiche in esattamente 6 classi. Osservando le figure 4.15d-4.19d si può solo intuire l'effettiva difficoltà di un *labeling* manuale. In una situazione realistica (non bidimensionale) non avremmo potuto visualizzare su un grafico il training set e i vettori peso dei neuroni e avremmo dovuto assolvere questo compito procedendo per tentativi. Vediamo come possiamo automatizzare questa fase con una *rete gerarchica*.

Mostriamo solo il caso di una *Neural-Gas gerarchica* a 3 strati con il primo strato composto da 50 neuroni, il secondo da 20 e il terzo da 6. Come mostra la figura 4.21, prima effettuiamo il *training* del primo strato sul *training set*, poi quello del secondo sui pesi del primo strato e poi quello del terzo sui pesi del secondo. Il terzo strato (figura 4.21c) ha quindi esattamente 6 neuroni, uno per ogni classe. La figura 4.21d mostra come lo spazio delle caratteristiche viene ripartito da questa rete. Utilizzando gli algoritmi ibridi del paragrafo 4.7 si ottiene pressappoco la stessa suddivisione ma con un tempo di esecuzione decisamente inferiore dato che ad apprendere è solo il primo strato di 50 neuroni.

Si potrebbe a questo punto obiettare che lo stesso risultato è ottenibile utilizzando direttamente una rete non gerarchica con 6 neuroni di output. Il problema è che una rete siffatta ha scarsissime *capacità di generalizzazione* e non sempre riesce a rappresentare adeguatamente il *training set*. Abbiamo comunque provato a far apprendere sul *training set* dei precedenti esempi anche una rete *Neural-Gas* con soli 6 neuroni output ottenendo una rappresentazione sufficiente (figure 4.22a-b) solo nell'85% dei casi. Nel restante 15% otteniamo la rappresentazione errata delle figure 4.22c-d dato che il cluster più grande è riuscito ad assorbire 2 neuroni. La configurazione finale dipende quindi fortemente dai valori casuali inizialmente attribuiti ai pesi. Di contro, le reti gerarchiche e ibride provate in precedenza, riescono nel 100% dei casi ed hanno inoltre un valore nettamente più basso della *distorsione media* dato che utilizzano, per la classificazione, 50 neuroni invece che 6. Avvicinando ulteriormente i due cluster già più vicini del training set (figura 4.23a), aumentano i fallimenti della rete a 6 neuroni (otteniamo la classificazione sbagliata di figura 4.23b nel 54% dei casi e quella giusta di figura 4.23c nel restante 46%) mentre sia la rete gerarchica che le reti ibride, continuano a classificare correttamente (figura 4.23d). Avvicinando ancora un altro cluster (figura 4.24a), sia la rete a 6 neuroni (figura 4.24b) che quella gerarchica (figure 4.24c) falliscono la rappresentazione. Per quanto riguarda le reti ibride, utilizzando la funzione di distanza di figura 4.14b (metodo del *centroide*), si ottengono gli stessi risultati della *Neural-Gas gerarchica* mentre utilizzando la funzione di figura 4.14a (metodo *nearest neighbour*) si ottiene la rappresentazione corretta di figura 4.24d.

Tutto ciò ci porta alla conclusione che, per evitare il *labeling*, bisogna preferire gli algoritmi gerarchici a quelli non gerarchici con pochi neuroni, dato che i primi sono più affidabili (data una distribuzione e un settaggio di parametri, o sbagliano sempre o non sbagliano mai la rappresentazione) rispetto ai secondi il cui comportamento risente troppo dei valori casuali inizialmente attribuiti ai pesi. La rete ibrida che utilizza il metodo del *centroide* è equivalente ad una rete gerarchica (ma molto più veloce) mentre quella che utilizza il metodo *nearest neighbour* se la cava meglio quando le

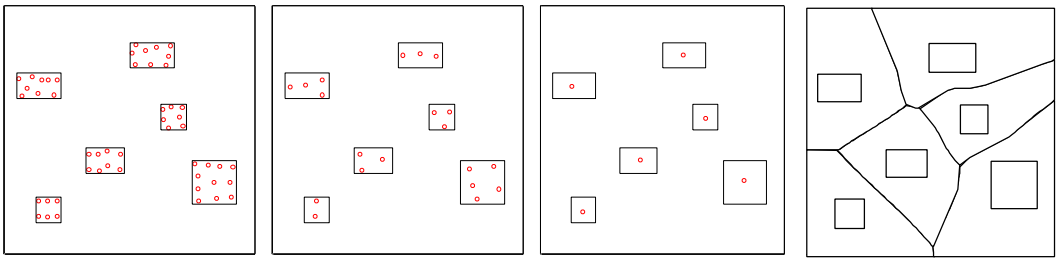

Fig. 4.21 - i 3 strati della ML-NeuralGas e la partizione del feature space.

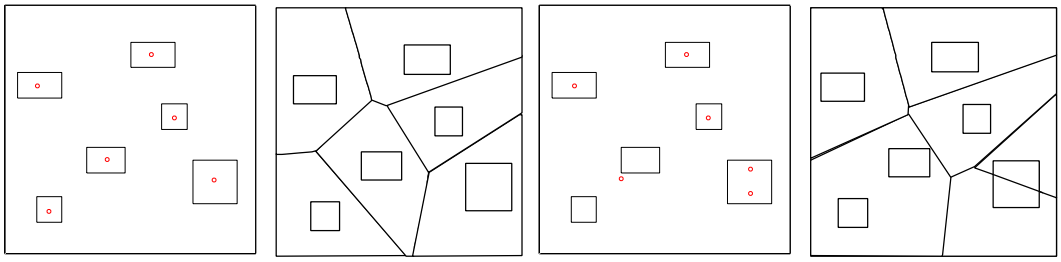

Fig. 4.22 - classificazioni giuste e sbagliate di una Neural-Gas a 6 output.

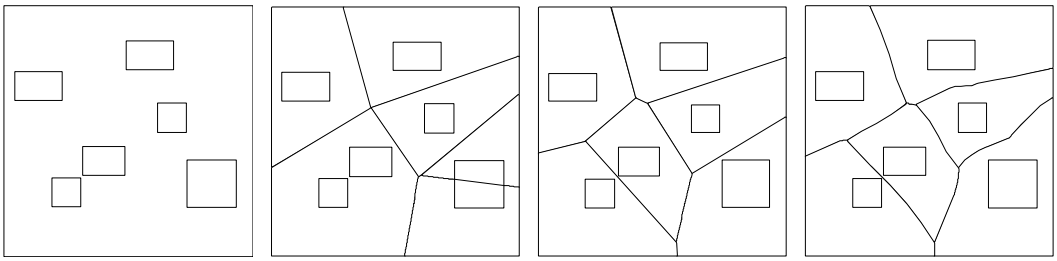

Fig. 4.23 - classificazioni giuste e sbagliate per il training set a sinistra.

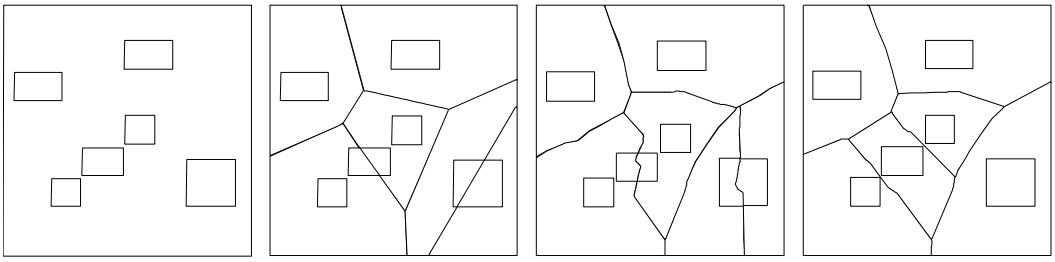

Fig. 4.24 - classificazioni giuste e sbagliate per il training set a sinistra.

classi sono più vicine o addirittura incastrate l'una dentro l'altra (vedi figura 4.25) anche se può generare problemi con classi parzialmente sovrapposte. In generale, quando in ogni classe è individuabile un centro attorno al quale si addensa la distribuzione dei punti, è preferibile una rete gerarchica (o il metodo del *centroide*), in caso contrario, si utilizzerà un algoritmo ibrido con il metodo *nearest neighbour*.

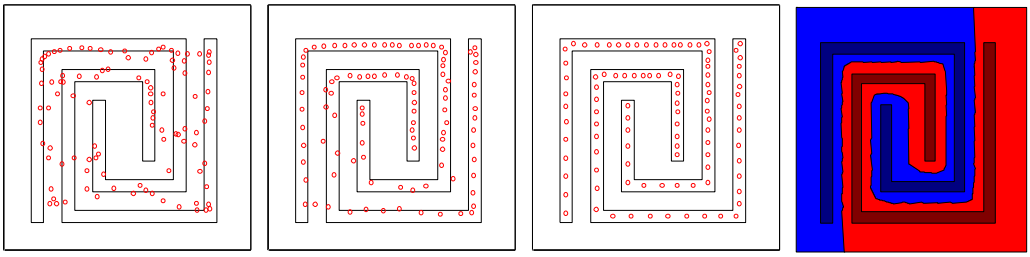

Fig. 4.25 - la Neural-Gas ibrida nearest neighbour in una situazione estrema.

Riferimenti.

R. Beale, T. Jackson "Neural Computing: An Introduction" *Institute of Physics Publishing Bristol and Philadelphia*, 1990.

B. Everitt "Cluster Analysis" *Social Science Research Council. Heinemann Educational Books. London*, 1977.

B. Fritzke "Growing Cell Structures - A Self-Organizing Network for Unsupervised and Supervised Learning" in *Neural Networks*, vol. 7, no. 9, pp. 1441-1460, 1994.

J. Koh, M. Suk, S. Bhandarkar "A Multilayer Self-Organizing Feature Map for Range Image Segmentation" in *Neural Networks*, vol. 8, no. 1, pp. 67-86, 1995.

T. Kohonen "Self-organized formation of topologically correct feature maps" in *Biological Cybernetics* vol. 43, pp. 59-69, 1982.

T. Martinetz, S. Berkovich, K. Schulten "Neural-Gas Network for Vector Quantization and its Application to Time-Series Prediction" in *IEEE transactions on Neural Networks*, vol. 4, no. 4, pp. 558-568, 1993.

CAPITOLO 5

Analisi delle Componenti Principali.

5.1 Introduzione.

Raramente si utilizzano, come input di una rete neurale, direttamente le caratteristiche ottenute in fase di *estrazione* ma spesso si rende necessaria una qualche trasformazione al fine di facilitare la classificazione dei pattern. Uno dei problemi che più frequentemente si devono risolvere è la diminuzione della dimensionalità dei pattern (del numero di caratteristiche) al fine di rendere più efficiente e più rapido il funzionamento delle reti.

Aumentare il numero di caratteristiche misurate sugli oggetti da classificare comporta, in generale, un miglioramento delle prestazioni della rete perché, intuitivamente, si hanno più informazioni a disposizione sulle quali basare l'apprendimento. In realtà ciò è vero solo fino ad un certo punto, superato il quale, le prestazioni della rete tendono a diminuire (si ottengono più classificazioni sbagliate). Ciò è dovuto al fatto che siamo costretti a lavorare su un insieme limitato di dati e quindi, aumentare la dimensionalità dello spazio dei pattern comporta un diradamento del nostro *training set* che diventerà una scarsa rappresentazione della distribuzione. Avremo di conseguenza bisogno di insiemi più ampi (la crescita deve essere esponenziale) che rallenteranno il processo di training e che porteranno miglioramenti infinitesimali. Questo problema è conosciuto in letteratura come *curse of dimensionality*. È meglio preferire una rete con pochi input perché questa ha meno parametri adattivi da determinare e quindi sono sufficienti anche piccoli insiemi di training. Creeremo in questo modo una rete più rapida e con maggiori capacità di generalizzazione. Il problema è ora quello di scegliere, tra le caratteristiche che abbiamo a disposizione, quelle da conservare e quelle da scartare, cercando di perdere la minor quantità possibile di informazione. In questo ci viene incontro la PCA.

La PCA (*Principal Component Analisys*) è una tecnica statistica il cui scopo è la riduzione della dimensione dei pattern e si basa sulla selezione delle caratteristiche più significative ovvero di quelle che portano più informazione. Essa è utilizzata in molti

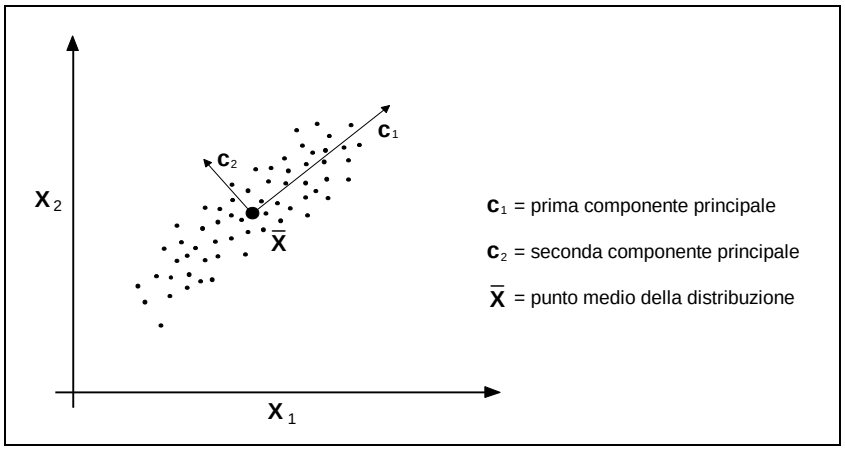

Fig. 5.1 - esempio di PCA applicata a dati bidimensionali.

campi e con nomi differenti: *espansione di Karhunen-Loeve*, *trasformazione Hotelling*, *approccio al sottospazio del segnale* ecc.

Data una distribuzione statistica di dati in uno spazio *L*-dimensionale, questa tecnica prende in esame le proprietà della distribuzione e cerca di determinare le componenti che massimizzano la *varianza* o che, in alternativa, minimizzano l'*errore di rappresentazione*. Queste componenti vengono dette *componenti principali* e sono combinazioni lineari di variabili casuali con la proprietà di massimizzare la varianza in relazione al valore degli *autovalori* (e quindi degli *autovettori*) della *matrice di covarianza* della distribuzione. Ad esempio, la prima *componente principale* è una combinazione lineare normalizzata che ha varianza massima, la seconda componete principale ha la seconda varianza massima e così via. Geometricamente la PCA corrisponde ad una rotazione degli assi coordinati in un nuovo sistema di coordinate tali che la proiezione dei punti sul primo asse abbia varianza massima, la proiezione sul secondo asse abbia la seconda massima varianza e così via (vedi figura 5.1). Grazie a quest'importante proprietà, questa tecnica ci permette di ridurre uno spazio di caratteristiche, preservando il più possibile le informazioni rilevanti.

Matematicamente, la PCA è definita come segue: consideriamo un vettore *L*-dimensionale x ottenuto da qualche distribuzione centrata intorno alla media $E(x) = 0$ e definiamo $C = E\left(xx^{\mathrm{T}}\right)$ la *matrice di covarianza* della distribuzione. L'*i*-esima *componente principale* di x è definita come $f_i^{\mathrm{T}} x$, dove f_i rappresenta l'autovettore normalizzato di *C* corrispondente all'*i*-esimo più grande autovalore l_i. Il sottospazio ottenuto dagli autovettori $f_1, \ldots, f_M$ con $M < L$, è chiamato il *sottospazio PCA* di dimensione *M*. Vedremo nei prossimi paragrafi metodi matematici e neurali per calcolare le *componenti principali* di una distribuzione.

5.2 Calcolo matematico delle componenti principali.

Consideriamo *N* punti in uno spazio *L*-dimensionale (che potrebbe essere uno *spazio di caratteristiche*) indicando ciascuno di essi con

$x_i = [x_{i1}, x_{i2}, \ldots, x_{iL}]^T \; i = 1, \ldots, N$ e supponiamo, senza perdita di generalità, che $E(x_i) = 0$ (E indica il *valor medio*). Possiamo rappresentare il generico vettore x_i come una combinazione lineare di un insieme di L vettori in questo modo:

$$x_i = \sum_{j=1}^{L} c_{ij} f_j \tag{5.1}$$

dove i c_{ij} sono coefficienti tali che $E(c_{ij}) = 0$ al variare di i, mentre i f_j sono vettori tra loro *ortonormali* ($f_i^T f_j = d_{ij}$) tali che $f_j = [f_{j1}, f_{j2}, \ldots, f_{jL}]^T \; j = 1, \ldots, L$. Se definiamo $c_i = [c_{i1}, c_{i2}, \ldots, c_{iL}]^T \; i = 1, \ldots, N$ allora l'equazione 5.1 può essere espressa in forma matriciale come: $x_i = f c_i$ dove f è la matrice che ha per colonne i vettori f_j. In questo caso l'espressione esplicita per i vettori c_i è: $c_i = f^T x_i$.

Supponiamo di voler ridurre la dimensionalità dello spazio da L a M con $M<L$ in modo da perdere la minor quantità possibile di informazione. Il primo passo consiste nel riscrivere la 5.1 in questo modo:

$$x_i = \sum_{j=1}^{M} c_{ij} f_j + \sum_{j=M+1}^{L} c_{ij} f_j . \tag{5.2}$$

per poi rimpiazzare tutti i c_{ij} (per $j = M+1, \ldots, L$) con delle costanti b_j così che ogni vettore x_i possa essere approssimato da un vettore $\bar{x}_i$ così definito:

$$\bar{x}_i = \sum_{j=1}^{M} c_{ij} f_j + \sum_{j=M+1}^{L} b_j f_j . \tag{5.3}$$

Otteniamo in questo modo una riduzione di dimensioni dato che la seconda sommatoria è costante e che quindi ogni vettore L-dimensionale x_i può essere espresso in maniera approssimata utilizzando un vettore M-dimensionale c_i. Vediamo ora come trovare i *vettori base* f_j ed i coefficienti b_j tali da minimizzare la perdita di informazioni. L'errore su x_i ottenuto dalla riduzione di dimensione è dato da:

$$x_i - \bar{x}_i = \sum_{j=M+1}^{L} (c_{ij} - b_j) \cdot f_j \tag{5.4}$$

possiamo quindi definire una funzione E_M che calcoli la somma dei quadrati degli errori in questo modo:

$$E_M = \frac{1}{2} \sum_{i=1}^{N} \| x_i - \bar{x} \|^2 = \frac{1}{2} \sum_{i=1}^{N} \sum_{j=M+1}^{L} (c_{ij} - b_j)^2 \tag{5.5}$$

dove abbiamo utilizzato la relazione di *ortonormalità*. Se poniamo la derivata prima di E_M rispetto a b_j pari a zero, otteniamo che:

$$b_j = \frac{1}{N} \sum_{i=1}^{N} c_{ij} = 0 \tag{5.6}$$

in virtù del fatto che abbiamo considerato c_{ij} tali che $E(c_{ij}) = 0$. La funzione di errore può essere quindi riscritta come:

$$E_M = \frac{1}{2}\sum_{j=M+1}^{L}\sum_{i=1}^{N} c_{ij}^2 = \frac{1}{2}\sum_{i=M+1}^{L} f_j^{\mathrm{T}}\left[\sum_{i=1}^{N} x_i x_i^{\mathrm{T}}\right] f_j = \frac{1}{2}\sum_{j=M+1}^{L} f_j^{\mathrm{T}} C f_j \qquad (5.7)$$

dove il primo passo segue dal fatto che $c_{ij} = f_j^{\mathrm{T}} x_i$ mentre la *C* che compare al secondo passo è la *matrice di covarianza* della distribuzione così definita:

$$C = \sum_{i=1}^{N}\left(x_i - E(x_i)\right)\left(x_i - E(x_i)\right)^{\mathrm{T}} = \sum_{i=1}^{N}(x_i x_i^{\mathrm{T}}) \qquad (5.8)$$

dato che abbiamo posto $E(x_i) = 0$. Non rimane ora che minimizzare la funzione E_M rispetto alla scelta dei *vettori base* f_j.

Può essere mostrato, ma non lo faremo in questa sede, che la scelta ottima si ha quando i *vettori base* soddisfano la condizione: $C f_j = l_j f_j$ per delle costanti l_j quindi, chiaramente, essi sono gli *autovettori* della matrice *C*. È da notare inoltre che, siccome la *matrice di covarianza* è reale e simmetrica, i suoi autovettori possono essere scelti ortonormali come richiesto. Tornando all'analisi della funzione di errore, notiamo che:

$$E_M = \frac{1}{2}\sum_{i=M+1}^{L} l_i \qquad (5.9)$$

quindi il minimo errore si ottiene scartando i più piccoli *L-M* autovalori ed i loro corrispondenti autovettori e conservando gli *M* più grandi che, normalizzati, andranno a costituire la matrice *f*. Il procedimento per calcolare le componenti principali è sintetizzato in figura 5.2.

5.3 Realizzazione neurale della PCA standard.

L'analisi delle componenti principali si può realizzare anche tramite una rete neurale in cui, i vettori peso dei neuroni convergono, durante la fase di apprendimento, agli

1. Si traslino i pattern $x_i \quad i = 1, \ldots, N$ in modo tale che $E(x_i) = 0$.

2. Si calcoli la matrice *C* a partire dall'insieme dei pattern x_i in questo modo:

$$C = \frac{1}{N}\sum_{i=1}^{N}(x_i x_i^{\mathrm{T}})$$

3. Si calcolino gli autovalori ed i corrispondenti autovettori di *C* normalizzando gli ultimi.

4. Si costruisca la matrice di trasformazione *f* con gli autovettori corrispondenti agli *M* autovalori più grandi. (Ogni colonna di *f* è un autovettore)

5. Si calcolino infine i *pattern immagine* di dimensione ridotta con la trasformazione lineare:

$$c_i = f^{\mathrm{T}} x_i$$

Fig. 5.2 - calcolo matematico delle componenti principali.

autovettori principali f_j per $j = 1, \ldots, M$. Tali reti hanno un apprendimento di tipo *hebbiano*: il valore di una connessione sinaptica in input ad un neurone viene cioè incrementato se e solo se l'input e l'output del neurone sono contemporaneamente attivi. Esse sono composte da uno strato di *L neuroni input* preposto a svolgere il solo compito di passaggio degli input allo strato successivo e da uno strato di *M neuroni output* totalmente connesso al precedente. I pesi di ogni neurone output formano un vettore peso *L*-dimensionale che rappresenta un autovettore. Esistono connessioni *feedback* durante l'apprendimento: se l'output del generico neurone giunge indistintamente come input a tutti i neuroni output siamo di fronte a una *rete simmetrica*; se invece esiste un ordine dei neuroni secondo il quale ogni neurone invia il suo output a se stesso ed ai neuroni con indici maggiori siamo di fronte ad una *rete gerarchica* (vedi figura 5.3). Dopo la fase di apprendimento queste connessioni (in rosso nella figura) vengono rimosse e la rete diventa puramente *feedforward*.

Abbiamo detto che questa rete realizza un *apprendimento hebbiano* (quindi non supervisionato). La legge di modificazione sinaptica non è però la regola *hebbiana* standard ovvero:

$$w_j^{(t+1)} = w_j^{(t)} + m \cdot y^{(t)} \cdot x_j^{(t)} \tag{5.10}$$

dove $x_j^{(t)}$, $w_j^{(t)}$ e $y^{(t)}$ sono, rispettivamente, il valore del *j*-esimo input, del *j*-esimo peso e dell'output della rete al tempo *t* (si suppone che la rete sia composta da un singolo neurone) mentre m rappresenta il *tasso di apprendimento*. L'applicazione diretta di questa regola renderebbe la rete instabile. Oja propose un altro tipo di regola per la modifica dei pesi al variare del tempo che trasforma la rete in un analizzatore di componenti principali. Egli pensò di normalizzare i vettori peso ad ogni passo e, partendo dalla 5.10 ottenne la seguente equazione:

$$w_j^{(t+1)} = w_j^{(t)} + m \cdot y^{(t)} \cdot \left[x_j^{(t)} - y^{(t)} w_j^{(t)} \right] \tag{5.11}$$

dove $m \cdot y^{(t)} \cdot x_j^{(t)}$ rappresenta il solito *incremento hebbiano* mentre $-y^{(t)} w_j^{(t)}$ è il termine stabilizzante che fa si che la sommatoria:

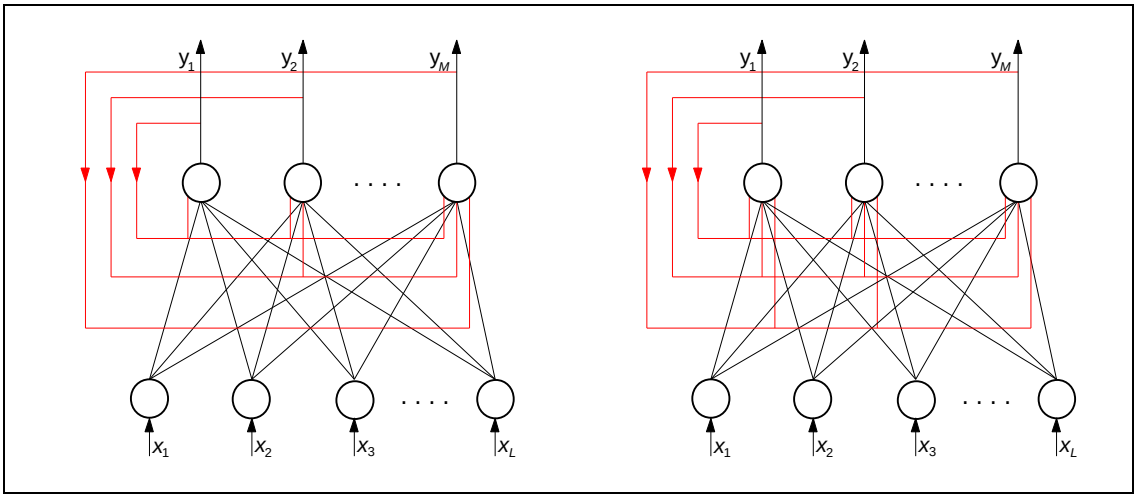

Fig. 5.3 - reti PCA gerarchica (a sinistra) e simmetrica (a destra).

$$\sum_{j=1}^{M}\left(w_j^{(t)}\right)^2 \tag{5.12}$$

sia limitata e vicina ad 1 senza che appaia una normalizzazione esplicita. La regola di Oja può essere generalizzata per reti che hanno più neuroni di output ottenendo i due algoritmi di figura 5.4 (algoritmo del sottospazio di Oja) e 5.5 (algoritmo di Sanger o *Generalized Hebbian Algoritm*). Il primo utilizza una rete simmetrica mentre il secondo una rete gerarchica. In entrambi gli algoritmi i vettori peso devono essere ortonormalizzati ovvero: $W^{\mathrm{T}}W = \mathrm{I}$.

La PCA emerge come soluzione ottima di diversi problemi di rappresentazione dell'informazione tra cui:

- *massimizzazione di varianze* soggette a trasformazioni lineari o degli output di una rete lineare sotto vincoli di ortonormalità;
- *minimizzazione dell'errore* quadratico medio quando i dati input sono approssimati usando un sottospazio lineare di dimensione inferiore;

1. Si inizializzino i vettori peso $w_i \quad i = 1, \ldots, M$ con valori piccoli e casuali. Si inizializzi inoltre la *soglia di apprendimento* ***eps*** e il *tasso di apprendimento m.* Si ponga *t*=0.

2. Si ortonormalizzi la matrice $W = (w_1, w_2, \ldots, w_L)$.

3. Si presenti in input il pattern: $x^{(t)} = (x_1, x_1, \ldots, x_L)$ e si calcoli l'output di ogni neurone in questo modo:

$$y_j^{(t)} = w_j^{(t)\mathrm{T}} x^{(t)}$$

4. Si adattino i pesi della rete utilizzando la seguente formula:

$$w_j^{(t+1)} = w_j^{(t)} + m \cdot y_j^{(t)}\left[x^{(t)} - \sum_{i=1}^{M} y_i^{(t)} w_i^{(t)} \right]$$

5. Si incrementi *t*. Se *t* non è multiplo di *N* (cardinalità del training set) si torni al **passo 3**.

6. Si effettui il *test della norma* in questo modo: si calcoli

$$tn = \sqrt{\sum_{j=1}^{M}\sum_{i=1}^{L}\left(w_{ji}^{(t)} - w_{ji}^{(t-N)} \right)^2}$$

se $tn \geq eps$ allora incrementa *t* e torna al **passo 2**, altrimenti **STOP**.

Fig. 5.4 - rete PCA standard: algoritmo del sottospazio di Oja.

Si consideri l'algoritmo di **figura 5.4** e si sostituisca il solo **passo 4** con il seguente:

4. Si adattino i pesi della rete utilizzando la seguente formula:

$$w_j^{(t+1)} = w_j^{(t)} + m \cdot y_j^{(t)}\left[x^{(t)} - \sum_{i=1}^{j} y_i^{(t)} w_i^{(t)} \right]$$

Fig. 5.5 - rete PCA standard: algoritmo GHA (Generalized Hebbian Algorithm).

- *non correlazione* degli output dopo una trasformazione ortonormale;
- *minimizzazione dell'entropia* di rappresentazione.

Nello stesso tempo la rete PCA ha alcune limitazioni che ne diminuiscono l'attrattiva ovvero:

- la rete è abile a realizzare solo corrispondenze input−output lineari;
- gli autovettori possono essere calcolati in maniera molto più efficiente usando tecniche matematiche;
- le componenti principali prendono in considerazione solo le covarianze dei dati che caratterizzano completamente solo distribuzioni Gaussiane;
- la rete non è in grado di separare sottosegnali indipendenti dalle loro combinazioni lineari.

Per queste ragioni è interessante studiare *generalizzazioni non lineari* della PCA ovvero algoritmi di apprendimento derivati dalla generalizzazione del problema di ottimizzazione della *PCA standard*. Essi possono essere divisi in due classi: gli algoritmi *PCA robusti* (paragrafi 5.4 e 5.5) e gli algoritmi *PCA non lineari* in senso stretto (paragrafo 5.6). Nei primi, il criterio da ottimizzare è caratterizzato da una funzione che cresce più lentamente della funzione quadratica e le condizioni iniziali sono le stesse della *PCA standard* (i vettori peso dei neuroni devono essere mutuamente ortonormali). In questi algoritmi la non linearità appare solo in determinati punti. Negli algoritmi *PCA non lineari*, invece, tutti gli output dei neuroni sono una funzione non lineare del responso. E' interessante inoltre notare che, mentre la *PCA standard*, per ottenere le componenti principali, ha bisogno di qualche forma di gerarchia per differenziare i neuroni output (l'algoritmo simmetrico ricava infatti solo combinazioni lineari delle componenti principali), nelle generalizzazioni non lineari, la gerarchia non è così importante dato che la funzione non lineare spezza la simmetria durante la fase di apprendimento.

5.4 Generalizzazione della massimizzazione della varianza.

Il problema standard quadratico che porta ad una soluzione PCA può essere ottenuto anche massimizzando le varianze degli output $E\left[y_i^{\mathrm{T}} y_i\right] = E\left[w_i^{\mathrm{T}} \cdot x \cdot x^{\mathrm{T}} \cdot w_i\right] = w_i^{\mathrm{T}} C w_i$ di una rete lineare sotto vincoli di ortonormalità. Questo problema non è ben definito finché i vettori peso *L*-dimensionali *w* dei neuroni non sono vincolati in qualche modo. In mancanza di conoscenze a priori, i vincoli di ortogonalità sono i più naturali perché permettono di misurare le varianze lungo direzioni che differiscono in modo massimale tra loro.

Se ci riferiamo a *reti gerarchiche*, l'*i*-esimo vettore peso w_i è vincolato ad avere norma unitaria e ad essere ortogonale ai vettori $w_j \;\; j = 1, \ldots, i-1$. Matematicamente ciò si può esprimere così: $w_i^{\mathrm{T}} w_j = \delta_{ij}$ per $j \le i$. Il vettore ottimo w_i sarà quindi l'*i*-esimo autovettore principale f_i della matrice di covarianza *C* e gli output della rete PCA

diventano le *componenti principali* dei vettori di dati. Lo stesso problema può essere risolto con *reti simmetriche* adottando il seguente vincolo: $w_i^{\mathrm{T}} w_j = \delta_{ij}$ per $j \neq i$. In forma matriciale si ha $W^{\mathrm{T}} W = \mathrm{I}$ dove $W = [w_1, w_2, \ldots, w_M]$ e I è la matrice unità. Se ora consideriamo l'output *y* della rete PCA lineare, il problema si può esprimere in forma compatta come massimizzazione di:

$$E(\|y\|^2) = traccia(W^{\mathrm{T}} C W). \tag{5.13}$$

La soluzione ottima è data, in tal caso, da una qualsiasi base ortonormale che spazzi il sottospazio PCA. Essa quindi non è unica. Il problema di massimizzazione della varianza sotto vincoli simmetrici di ortonormalità porta quindi a reti simmetriche, le cosiddette *reti del sottospazio* PCA.

Consideriamo ora la generalizzazione del problema di massimizzazione della varianza per la ***PCA robusta***. Invece di usare il solito valore quadratico medio definito precedentemente, possiamo massimizzare una media più generale ovvero:

$$E\left[f(x^{\mathrm{T}} w_i) \right]. \tag{5.14}$$

La funzione $f(t)$ deve essere una valida funzione costo che cresce più lentamente del quadrato, almeno per grandi valori di *t*. In particolare ipotizziamo che $f(t)$ sia pari, non negativa, quasi ovunque continua, differenziabile e che $f(t) \leq t^2/2$ per grandi valori di $|t|$. Inoltre il suo unico minimo è raggiunto per $t=0$ e $f(t_1) \leq f(t_2)$ se $|t_1| \leq |t_2|$. Valide funzioni costo sono : $\ln(\cosh(at)), \tanh^2(at)$ dove a rappresenta un fattore di scalaggio che dipende dal range entro cui variano i valori input. In tal caso il criterio da massimizzare, per ogni vettore peso w_i, è:

$$J(w_i) = E\left[f(x^{\mathrm{T}} w_i) \right] + \sum_{j=1}^{l(i)} \lambda_{ij} \left[w_i^{\mathrm{T}} w_j - \delta_{ij} \right] \tag{5.15}$$

Nella sommatoria, i coefficienti di Lagrange impongono i necessari vincoli di ortonormalità.

Sia il problema gerarchico, sia quello simmetrico, possono essere discussi sotto il criterio generale *J*. Nel caso simmetrico standard il limite superiore dell'indice di sommatoria è $l(i)=M$; nel caso gerarchico invece è $l(i)=i$. Il vettore peso ottimo dell'*i*-esimo neurone definisce allora la componente robusta dell'*i*-esimo autovettore principale f_i. Il *gradiente* di $J(w_i)$ rispetto a w_i è:

$$h(i) = \frac{\partial J(w_i)}{\partial w_i} = E\left[x g(x^{\mathrm{T}} w_i) \right] + 2\, \lambda_{ii} w_i + \sum_{j=1, j \neq i}^{l(i)} \lambda_{ij} w_j \tag{5.16}$$

dove $g(t)$ è la derivata $\partial f(t) / \partial t$ di $f(t)$. All'ottimo il gradiente deve essere nullo per $i = 1, \ldots, M$. Inoltre la differenziazione dei coefficienti di Lagrange porta ai vincoli di ortonormalità $w_i^{\mathrm{T}} w_j = \delta_{ij}$.

Un algoritmo a gradiente discendente atto a massimizzare l'equazione 5.14 si ottiene inserendo la stima $h(i)$ del vettore gradiente (equazione 5.16) al passo dell'aggiornamento dei pesi che diventa:

Si consideri l'algoritmo di **figura 5.4** e si sostituisca il solo **passo 4** con il seguente:

4. Si adattino i pesi della rete utilizzando la seguente formula:

$$w_j^{(t+1)} = w_j^{(t)} + m \cdot g\left(y_i^{(t)}\right) \cdot e_i^{(t)} \quad \text{dove} \quad e_j^{(t)} = x^{(t)} - \sum_{i=1}^{l(j)} y_i^{(t)} w_i^{(t)}$$

e, inoltre, $l(i)=i$ nel caso gerarchico mentre $l(i)=M$ nel caso simmetrico.

Fig. 5.6 - algoritmo PCA robusto di gen. della massimizzazione della varianza.

$$w_i^{(t+1)} = w_i^{(t)} + m \cdot h^{(t)}(i) \tag{5.17}$$

Per ottenere le stime del *gradiente istantaneo standard*, semplicemente si omettono i valori medi e si usano invece i valori istantanei delle quantità in questione. L'aggiornamento diventa quindi:

$$w_i^{(t+1)} = w_i^{(t)} + m \cdot \left[\mathrm{I} - \sum_{j=1}^{l(i)} w_j^{(t)} w_j^{(t)\mathrm{T}} \right] \cdot x^{(t)} \cdot g\left(x^{(t)\mathrm{T}} w_i^{(t)}\right). \tag{5.18}$$

Riconsiderando la funzione costo, le ipotesi fatte su essa implicano che la sua derivata $g(t)$ sia una funzione dispari non decrescente di t. Per ragioni di stabilità è richiesto che valgano almeno: $g(t) \le 0$ per $t<0$ o $g(t) \le 0$ per $t>0$. Se definiamo il *vettore errore di rappresentazione istantaneo* come:

$$e_i^{(t)} = x^{(t)} - \sum_{j=1}^{l(i)} \left(x^{(t)\mathrm{T}} w_j^{(t)} \right) \cdot w_j^{(t)} = x^{(t)} - \sum_{j=1}^{l(i)} y_j^{(t)} w_j^{(t)} \tag{5.19}$$

possiamo sintetizzare il passo di aggiornamento dei pesi (equazione 5.18) come in figura 5.6. È da notare che, dato che nel caso simmetrico $e_i(k)$ è uguale per tutti i neuroni, la 5.18 può anche essere espressa in forma matriciale come:

$$W^{(t+1)} = W^{(t)} + m \cdot \left(\mathrm{I} - W^{(t)} W^{(t)\mathrm{T}}\right) \cdot x \cdot g\left(x^{\mathrm{T}} W^{(t)}\right) = W^{(t)} + m \cdot e^{(t)} \cdot g\left(y^{(t)\mathrm{T}}\right). \tag{5.20}$$

E' interessante notare che la soluzione ottima per il criterio robusto in generale non coincide con la soluzione standard ma è molto vicina a questa. Ad esempio se consideriamo $f(t)=|t|$, le direzioni w_i che massimizzano $E\left[|x^{\mathrm{T}} w_i|\right]$ sono, per qualche arbitraria distribuzione non simmetrica, differenti dalle direzioni che massimizzano la varianza $E\left[(x^{\mathrm{T}} w_i)^2\right]$ sotto condizioni di ortonormalità.

5.5 Generalizzazione della minimizzazione dell'errore.

Consideriamo l'approssimazione lineare $\bar{x}$ dei vettori x in termini di un insieme di vettori w_j per $j = 1, \ldots, l(i)$. Poiché il numero $l(i)$ di vettori base w_j è solitamente minore della dimensione L dei vettori dati, ci sarà un certo errore detto *errore istantaneo di rappresentazione* $e_i^{(t)} = x_i^{(t)} - \bar{x}_i^{(t)}$ per ogni vettore $x^{(t)}$. Le soluzioni della *PCA standard* sono ottenute minimizzando il quadrato di questo errore ovvero la

quantità: $E\left[\|e_i\|^2\right] = E\left[\|x_i - \bar{x}_i\|^2\right]$. Vediamo ora come effettuare la *generalizzazione robusta* dell'*errore quadratico medio di rappresentazione*. Algoritmi *PCA robusti* si possono ottenere minimizzando il criterio:

$$J(e_i) = 1^{\mathrm{T}} E\left[f(e_i)\right] \tag{5.21}$$

dove il vettore *L*-dimensionale 1 ed $f(t)$ soddisfano le ipotesi prima menzionate. Minimizzando la 5.21 rispetto a w otteniamo l'algoritmo a gradiente discendente mostrato in figura 5.7. L'algoritmo può essere applicato, come al solito, sia al caso simmetrico che gerarchico ma nel caso simmetrico $l(i)=M$ e quindi:

$$W^{(t+1)} = W^{(t)} + m \cdot \left(x^{(t)} \cdot g\left(e^{(t)\mathrm{T}}\right) \cdot W^{(t)} + g(e^{(t)}) \cdot x^{(t)\mathrm{T}} \cdot W^{(t)} \right) \tag{5.22}$$

Il primo termine $w_j^{(t)\mathrm{T}} \cdot g\left(e^{(t)}\right) \cdot x^{(t)}$ nell'equazione di figura 5.7 è proporzionale ad x per tutti i vettori peso, inoltre possiamo ipotizzare che il *valor medio* del coefficiente $w_j^{(t)\mathrm{T}} \cdot g\left(e^{(t)}\right)$ sia prossimo a zero perché il vettore di errore $e^{(t)}$ deve essere relativamente piccolo dopo una convergenza iniziale. Questo termine può quindi essere trascurato senza commettere un grande errore, il che ci porta all'algoritmo di figura 5.8.

Confrontando gli algoritmi di figura 5.8 (robusto approssimato di generalizzazione della minimizzazione dell'errore) e di figura 5.6 (robusto di generalizzazione della massimizzazione della varianza) notiamo che essi, pur essendo derivati da due differenti criteri di ottimizzazione, sono molto simili. La sola differenza che salta all'occhio consiste nel fatto che la funzione non lineare $g(t)$ è applicata all'errore $e^{(t)}$ nel primo caso e all'output $y^{(t)}$ nel secondo. Questo ha un'importante conseguenza: se la rete apprende attraverso l'algoritmo di figura 5.8, la corrispondenza input-output finale sarà ancora lineare, se si utilizza l'algoritmo di figura 5.6, ciò non avviene dato

Si consideri l'algoritmo di **figura 5.4** e si sostituisca il solo **passo 4** con il seguente:

4. Si adattino i pesi della rete utilizzando la seguente formula:

$$w_j^{(t+1)} = w_j^{(t)} + m \cdot \left(w_j^{(t)\mathrm{T}} \cdot g\left(e_i^{(t)}\right) \cdot x^{(t)} + x^{(t)\mathrm{T}} \cdot w_j^{(t)} \cdot g\left(e_j^{(t)}\right) \right) \text{ dove } e_j^{(t)} = x^{(t)} - \sum_{i=1}^{l(j)} y_i^{(t)} w_i^{(t)}$$

e, inoltre, $l(i)=i$ nel caso gerarchico mentre $l(i)=M$ nel caso simmetrico.

Fig. 5.7 - algoritmo PCA robusto di generalizzazione della minimizzazione d'errore.

Si consideri l'algoritmo di **figura 5.4** e si sostituisca il solo **passo 4** con il seguente:

4. Si adattino i pesi della rete utilizzando la seguente formula:

$$w_j^{(t+1)} = w_j^{(t)} + m \cdot y_j^{(t)} \cdot g\left(e_j^{(t)}\right) \text{ dove } e_j^{(t)} = x^{(t)} - \sum_{i=1}^{l(j)} y_i^{(t)} w_i^{(t)}$$

e, inoltre, $l(i)=i$ nel caso gerarchico mentre $l(i)=M$ nel caso simmetrico.

Fig. 5.8 - algoritmo PCA robusto approssimato di gen. della minimizzazione d'errore.

Si consideri l'algoritmo di **figura 5.4** e si sostituisca il solo **passo 4** con il seguente:

4. Si adattino i pesi della rete utilizzando la seguente formula:

$$w_j^{(t+1)} = w_j^{(t)} + m \cdot g(y_j^{(t)}) \cdot \left(b_j^{(t)}\right) \text{ dove } b_j^{(t)} = x^{(t)} - \sum_{i=1}^{l(j)} g\left(y_i^{(t)}\right) w_i^{(t)}$$

e, inoltre, $l(i)=i$ nel caso gerarchico mentre $l(i)=M$ nel caso simmetrico.

Fig. 5.9 - algoritmo PCA non lineare.

che gli output della rete sono non lineari: $g\left(x^{(t)\mathrm{T}} w_i^{(t)}\right)$.

5.6 PCA non lineare.

Consideriamo ora la versione non lineare della PCA. Un modo euristico di fare questo è richiedere che gli output dei neuroni siano sempre non lineari e che siano: $g(y_i) = g(w_i^{\mathrm{T}} x)$. Applicando ciò all'equazione di figura 5.6 arriviamo alla seguente regola di adattamento pesi:

$$w_j^{(t+1)} = w_j^{(t)} + m \cdot g\left(y_j^{(t)}\right) \cdot b_j^{(t)} \tag{5.23}$$

che è simile alla precedente tranne che ora il vettore di errore è definito come:

$$b_j^{(t)} = x^{(t)} - \sum_{i=1}^{l(j)} g\left(y_i^{(t)}\right) w_i^{(t)} \,. \tag{5.24}$$

Tutto ciò è sintetizzato in figura 5.9. Nel caso simmetrico l'algoritmo deriva direttamente dalla generalizzazione dell'algoritmo del sottospazio PCA di Oja e può essere espresso in forma matriciale come:

$$W^{(t+1)} = W^{(t)} + m \cdot b^{(t)} \cdot g\left(y^{(t)\mathrm{T}}\right) \tag{5.25}$$

Il vantaggio maggiore di questa rete sembra essere il fatto che i coefficienti non lineari implicitamente prendono in considerazione informazioni statistiche di grado superiore al secondo e gli output divengono più indipendenti rispetto alle reti PCA standard.

Riferimenti.

C. M. Bishop "Neural Networks for Pattren Recognition" *Oxford University Press*, 1995.

J. Karhunen, J. Joutsensalo "Generalizations of Principal Component Analysis, Optimization Problems and Neural Networks" in *Neural Networks* vol. 8, no. 4, pp. 549-562, 1995.

E. Oja, J. Karhunen, L. Wang, R. Vigario "Principal and Independent Components in Neural Networks - Recent Developments" al *7th Italian Workshop on Neural Networks*,1995.

E. Oja, H. Ogawa, J. Wangviwattana "Learning in Nonlinear Contrained Hebbian Networks" in *Artificial Neural Networks* pp. 385-390, 1991.

CAPITOLO 6

Primo Passo: Segmentazione

6.1 Introduzione.

La segmentazione è un processo di ripartizione di un'immagine in regioni sconnesse dove ogni regione è omogenea e l'unione di regioni adiacenti è non omogenea. Per un'immagine astronomica, la segmentazione è il mezzo tramite il quale si individuano tutti i corpi celesti in esso contenuti, evidenziandone le caratteristiche in modo più netto e filtrando i contributi relativi al rumore. E' immediato, pertanto, individuarne l'utilità nel dominio di questo lavoro di tesi dato che, prima di poter classificare un qualsiasi oggetto, bisogna identificarlo all'interno del campo in cui si trova.

Daremo una definizione più formale di segmentazione: sia *F* l'insieme di tutti i pixel che compongono l'immagine da segmentare e sia un predicato di uniformità (o omogeneità) definito su gruppi di pixel connessi; una segmentazione è una partizione di *F* in una famiglia di sottoinsiemi $(S_1, S_2, ..., S_n)$ tali che:

$$\bigcup_{i=1}^{n} S_i = F \quad \text{con} \quad S_i \cap S_j = \varnothing, \quad i \neq j$$

dove il predicato di uniformità $P(S_i) = vero$ per ogni *i* e $P(S_i \cup S_j) = falso$ quando S_i è adiacente a S_j.

Centinaia di tecniche di segmentazione sono presenti in letteratura, ma non c'è un singolo metodo che si possa considerare buono per tutte le immagini, né ci sono particolari immagini per le quali tutti i metodi siano ugualmente buoni. Tra tutti i metodi esistenti però, le tecniche che si basano sull'utilizzo di ***reti neurali non supervisionate*** sono quelle che meglio lavorano in ambienti rumorosi. Abbiamo visto nei capitoli 2 e 4 che le reti neurali lavorano come classificatori e, in particolare, che le reti non supervisionate svolgono compiti di clustering. Un problema di segmentazione può tradursi facilmente in un problema di classificazione se si definiscono delle classi sui pixel e si riscrive il predicato di uniformità in modo tale che $P(S_i) = vero$ se e solo se tutti i pixel di S_i appartengono alla stessa classe. Il problema diventa quindi quello di individuare la classe di ogni pixel dell'immagine e si può risolvere con reti neurali

non supervisionate che genereranno le classi in modo da massimizzarne la distanza. Utilizzeremo quindi le reti del capitolo 4 con particolare riferimento alle reti gerarchiche e ibride che ci risparmieranno il *labeling* manuale dei neuroni di output. Utilizzando, inoltre, le tecniche di *analisi delle componenti principali* del capitolo 5, potremo lavorare agevolmente sia su una sola banda, sia su tre bande di colore simultaneamente.

6.2 Segmentazione di un campo stellare.

Come è strato già accennato nel paragrafo 1.4, scopo della fase di segmentazione è la costruzione, a partire da una lastra digitalizzata di un campo stellare, di una nuova immagine in due colori detta *maschera*, dove ogni pixel è azzerato se il corrispondente pixel sulla lastra fa parte dello sfondo ed è invece posto uguale a uno se appartiene ad un oggetto. La *maschera* sarà, insieme all'immagine sorgente, l'input della fase successiva: la *valutazione degli oggetti*. Per creare questa *maschera* utilizziamo una rete neurale non supervisionata che determina la classe (sfondo/oggetto = 0/1) di ogni pixel dell'immagine. Ogni pixel dovrà quindi essere rappresentato da un *pattern*.

Potremmo utilizzare, allo scopo, il solo valore del pixel che, detto per inciso, corrisponde all'intensità della luminosità rilevata in quel punto dal telescopio. Per ottenere una classificazione che tenga conto del rumore, è però necessario considerare anche il contesto in cui il pixel si trova, dato che un pixel molto luminoso in un contesto di pixel scuri o un pixel scuro in un contesto luminoso sono quasi certamente errori di rilevamento dovuti a cause molteplici quali: turbolenze atmosferiche, fenomeni di diffrazione e aberrazione provocati dal sistema ottico, errori di messa a fuoco, polvere, difetti del filtro o dispersioni di luce nell'emulsione. Un pixel luminoso in un contesto scuro è quindi fondo, viceversa, un pixel scuro in un contesto luminoso è oggetto. Per tener conto del contesto in cui un pixel si colloca, consideriamo quindi non il suo solo valore, ma il valore di tutti i pixel in una finestra $n \times n$ in esso centrata. I

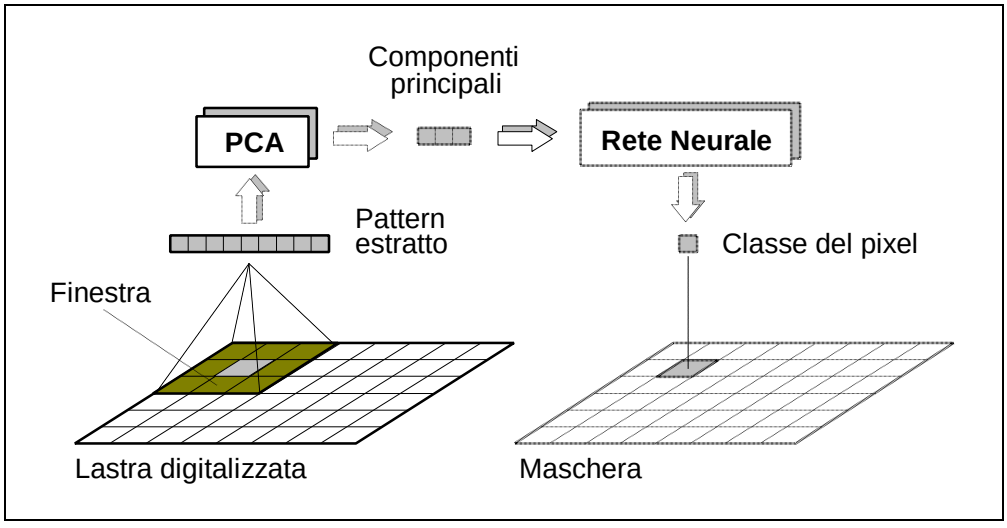

Fig. 6.1 - schema di funzionamento della segmentazione su una banda.

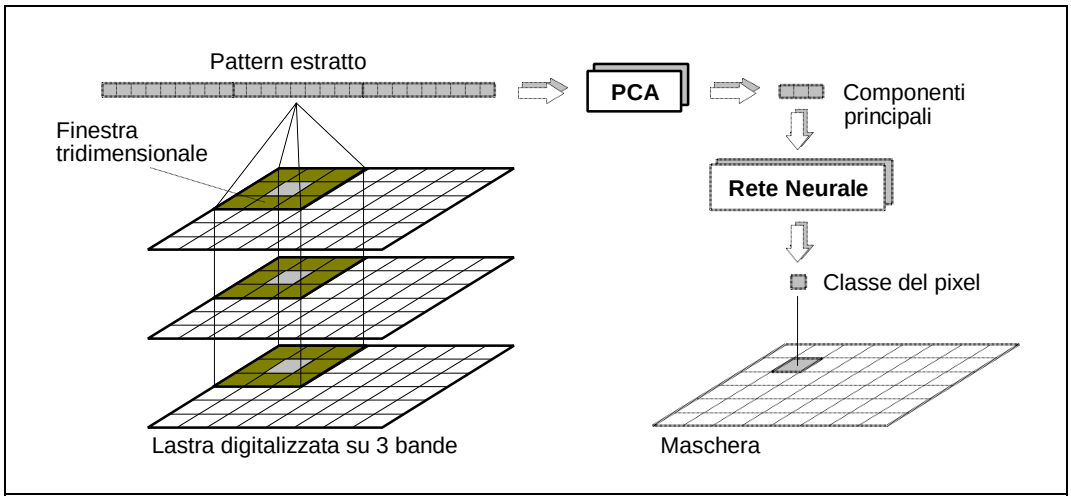

Fig. 6.2 - schema di funzionamento della segmentazione su tre bande.

valori di tutti questi pixel costituiscono il *pattern* corrispondente all'oggetto-pixel osservato e, su questi *pattern*, verrebbe fatta operare la *rete neurale* se non fosse che i *pattern* costruiti in questo modo hanno un numero di componenti eccessivo per poter essere agevolmente trattati dalla rete. Siamo costretti quindi a decrementare la dimensionalità dei *pattern* utilizzando le tecniche di PCA illustrate nel capitolo 5. Le caratteristiche saranno ridotte, in questo modo a 3 o 4 componenti principali a vantaggio sia del tempo di esecuzione del training della rete si di quello di vero e proprio utilizzo.

Vediamo ora come avviene, passo per passo, la segmentazione. Si parte con una lastra digitalizzata su una banda (vedi figura 6.1) o su tre bande (vedi figura 6.2) e si fa scorrere su di essa una finestra $n \times n$ (nelle figure n=3) da sinistra verso destra e dall'alto verso il basso. Ad ogni iterazione si costruisce il *pattern* $b \times n \times n$-dimensionale (dove b è il numero di bande) con i pixel di questa finestra; si calcolano le componenti principali tramite una delle reti PCA viste nel capitolo 5; si utilizzano le componenti principali come input della rete neurale che deciderà la classe del pixel centrale della finestra e lo andrà ad attivare o disattivare sulla maschera. Alla fine del procedimento, tutti i pixel della maschera saranno correttamente settati a 0 o a 1 tranne quelli periferici distanti meno di (n-1)/2 pixel dal bordo che verranno, per convenzione, posti a zero. La perdita di informazione sul bordo non è dannosa dato che, come abbiamo visto nel capitolo 1, le lastre che abbiamo a disposizione sono parzialmente sovrapposte. L'informazione che si perde sul bordo di una lastra è quindi presente su altre lastre in zone non periferiche.

Prima di passare ad illustrare gli esperimenti di segmentazione che abbiamo effettuato allo scopo di stabilire gli algoritmi e i parametri che offrono risultati migliori, è necessario spendere qualche parola sul ***training*** delle reti (PCA e di classificazione). Il *training* delle reti PCA viene effettuato sui *pattern* contenuti in zone significative dell'immagine sorgente stabilite dall'utente in maniera interattiva. E' necessario saper scegliere, in questa fase, dei *pattern* rappresentativi, di modo che la rete riesca a generalizzare sui *pattern* del *training set* e, di conseguenza, a segmentare l'immagine nel miglior modo possibile. Una volta che la rete PCA ha appreso, si

utilizzeranno gli autovettori da essa calcolati, per estrarre le componenti principali del *training set* e darli in input (ancora come *training*) alla rete di classificazione. Dopo questa seconda fase, le reti sono pronte ad essere utilizzate. La scelta delle zone significative deve essere fatta una volta per ogni lastra 32000×32000 ed è valida per tutti i *footprint* estratti dalla lastra. Purtroppo non è stato possibile generare un *training set* generale che desse luogo a reti capaci di segmentare qualsiasi lastra dato che purtroppo, le lastre, sono diversissime l'una dall'altra essendo state ottenute in notti con diversa visibilità oltre che digitalizzate e calibrate con tipi diversi di hardware.

6.3 Esperimenti di analisi delle componenti principali.

Un primo parametro da fissare prima di procedere a successive indagini circa l'efficienza degli algoritmi utilizzati, è la dimensione della finestra che si fa scorrere sull'immagine. A seguito di numerosi esperimenti effettuati su tipi differenti di lastre, abbiamo stabilito che la dimensione ottimale della finestra è di 5×5 pixel. Finestre più piccole danno luogo a segmentazioni rumorose, mentre finestre più grandi rallentano la fase di analisi delle componenti principali senza migliorare in maniera apprezzabile il risultato. Ciò premesso, passiamo ad illustrare gli esperimenti fatti per individuare il tipo di PCA con le prestazioni migliori.

Abbiamo selezionato tipi differenti di lastre e, dopo aver fissato su ognuna di queste alcune finestre significative, abbiamo fatto funzionare gli algoritmi PCA del capitolo 5 per ottenerne le componenti principali. Le prestazioni relative alle velocità medie di convergenza per un *training set* di 10000 *pattern* e una soglia di apprendimento pari a 1×10^{-5} sono riportate nella tabella di figura 6.3. Come si vede, le reti più veloci a convergere sono risultate le PCA lineari (GHA e sottospazio di Oja) e, subito dopo, le PCA ottenute dalla generalizzazione della massimizzazone della varianza.

La velocità però non è stato l'unico criterio di valutazione e neppure il più importante. Per la scelta del metodo più efficace, particolare rilevanza è stata data allo studio delle segmentazioni ottenute fissando una rete neurale non supervisionata e variando l'algoritmo PCA. Ci siamo così resi conto che le reti PCA potevano essere suddivise in due grandi classi: reti che hanno una corrispondenza input-output lineare

	Tipo di rete		**n. cicli**	**tempo (sec.)**
1	GHA (*Generalized Hebbian Algorithm*)		45	60
2	Sottospazio di Oja		42	64
3	Gerarchica robusta	massimizz. varianza	77	109
4	" "	minimizz. errore	97	215
5	" "	approssimata	97	177
6	Gerarchica non lineare		142	199
7	Simmetrica robusta	massimizz. varianza	87	127
8	" "	minimizz. Errore	107	343
9	" "	approssimata	111	306
10	Simmetrica non lineare		143	196

Fig. 6.3 - prestazioni delle reti PCA.

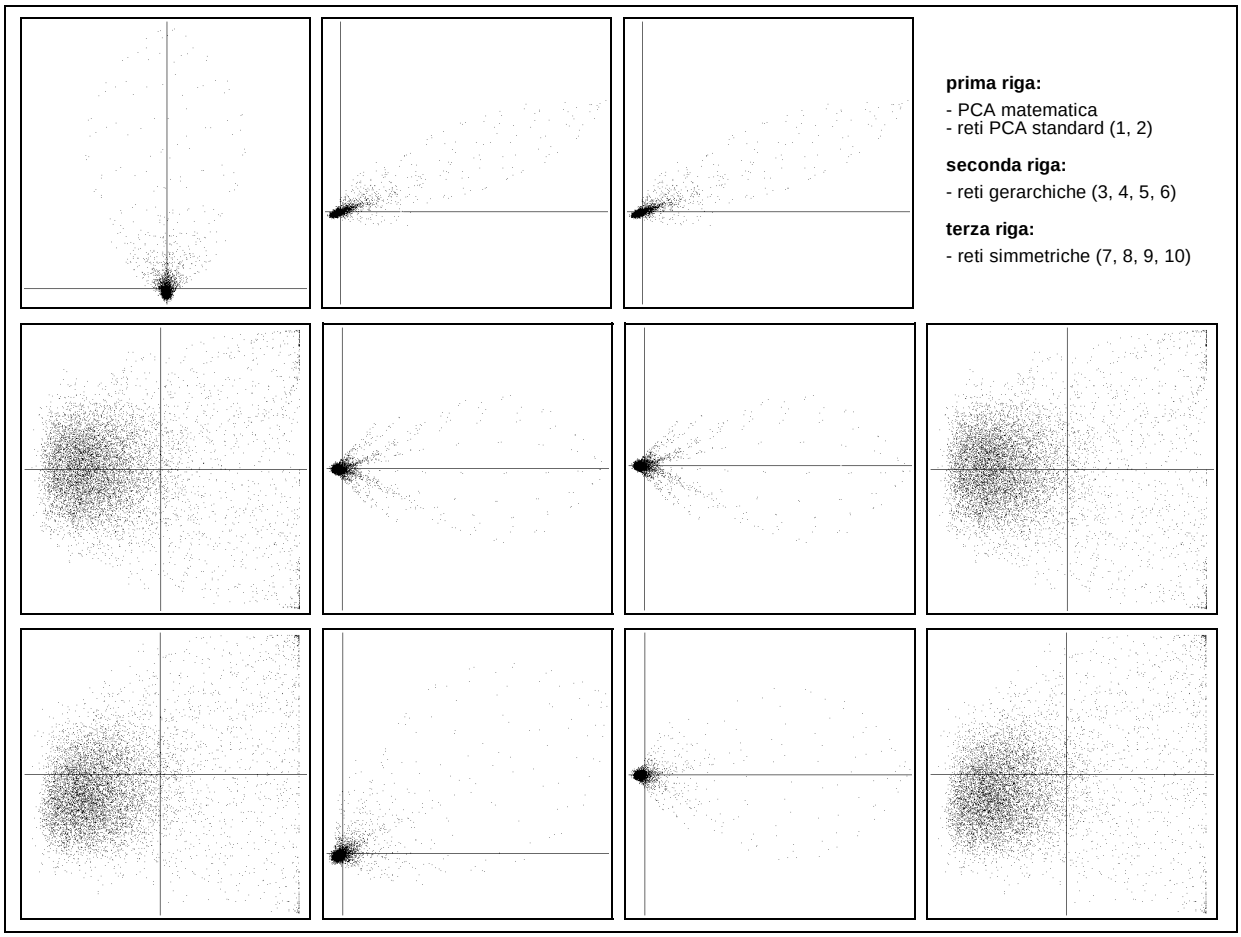

Fig. 6.4 - distribuzioni ottenute da diversi algoritmi PCA con due componenti principali.

(reti di *primo tipo*) e reti che hanno una corrispondenza input-output non lineare (reti di *secondo tipo*). Le reti di *primo tipo* (reti n. 1, 2, 4, 5, 8 e 9) generano una *maschera* che individua solo corpi celesti di magnitudine medio-bassa (oggetti molto luminosi) confondendo gli oggetti più deboli con il fondo. Le reti del *secondo tipo* (reti n. 3, 6, 7 e 10) permettono invece, sotto certe condizioni, di individuare anche oggetti ad alta magnitudine. La condizione per ottenere questo risultato è l'utilizzo, negli algoritmi delle figure 5.6 e 5.9, di una funzione $g(ay)$ di tipo sigmoidale[1] (tengente iperbolica). Per capire le motivazioni di fondo di questa scelta, occorre studiare le distribuzioni delle proiezioni dei punti di *training* dopo l'applicazione della PCA.

Come mostra la figura 6.4 (dove abbiamo considerato solo due componenti principali), le reti di *primo tipo* danno luogo a distribuzioni caratterizzate da un *nucleo* piccolo e molto fitto e da una serie di *punti sparsi* lontani tra loro. Il *nucleo* è costituito sia dalle proiezioni dei punti di fondo, sia da quelle dei punti appartenenti a oggetti poco luminosi; i *punti sparsi* sono invece proiezioni di punti appartenenti ad oggetti a bassa magnitudine (confronta con la figura 6.5). Gli oggetti molto luminosi, se presenti nel training set, finiscono quindi con sminuire l'importanza degli oggetti più deboli

[1] α è un fattore di scaling variabile a seconda dell'ampiezza del range di definizione della luminosità. Per lastre a 16 bit per pixel (come quelle utilizzate dal progetto CRoNaRio) il valore ottimale per α è 2×10^{-4}

schiacciando questi ultimi sul nucleo. Utilizzando reti non supervisionate per classificare la distribuzione ottenuta, tutti i punti del nucleo saranno, per forza di cose, inseriti nello stesso cluster (sono molto vocini tra loro) che sarà individuato come fondo avendo i punti a più bassa luminosità. Una prima soluzione al problema è di evitare di utilizzare oggetti molto luminosi per addestrare la rete. Non vogliamo però che il nostro programma risulti troppo sensibile al training set, dato che quest'ultimo deve essere indicato dall'utente in maniera interattiva e non sempre l'utente è in grado di effettuare scelte ottimali.

Utilizzando le reti del *secondo tipo* con funzioni di tipo sigmoidale, otteniamo invece distribuzioni con un nucleo meno denso che copre maggiormente la superficie interessata. Come mostra la figura 6.5, questo tipo di rete dà un maggior rilievo ai punti degli oggetti deboli staccandoli dal fondo. Per capire come ciò avviene, consideriamo l'esempio mostrato in figura 6.6.

Supponiamo di avere una prima distribuzione di punti (figura 6.6a) ottenuti estraendo le caratteristiche di pixel di fondo (punti neri), di corpi celesti con alta magnitudine (punti rossi) e di corpi celesti con bassa magnitudine (punti blu). Utilizzando reti non supervisionate per il *clustering* e dovendo distinguere solo due gruppi (fondo/oggetto), punti neri e rossi verranno classificati come un primo *cluster* (fondo) e i restanti come un secondo *cluster* (oggetto). Le reti non possono fare altrimenti poiché i valori dei punti blu sono molto grandi rispetto agli altri. Applicando però una funzione sigmodale alla distribuzione (figura 6.6b) i punti blu vengono compattati grazie al fatto di appartenere alla zona più satura della funzione. In questo modo otteniamo una nuova distribuzione (figura 6.6c) che accentua la differenza tra punti rossi e neri e che comporta un clustering più sensibile agli oggetti deboli.

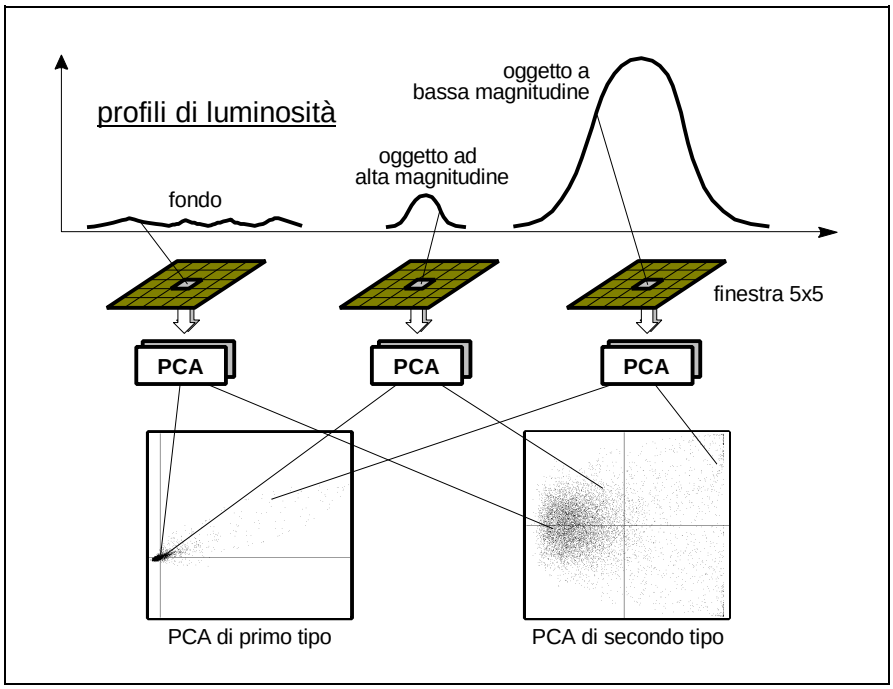

Fig. 6.5 - localizzazione delle proiezioni dei pattern nello spazio PCA.

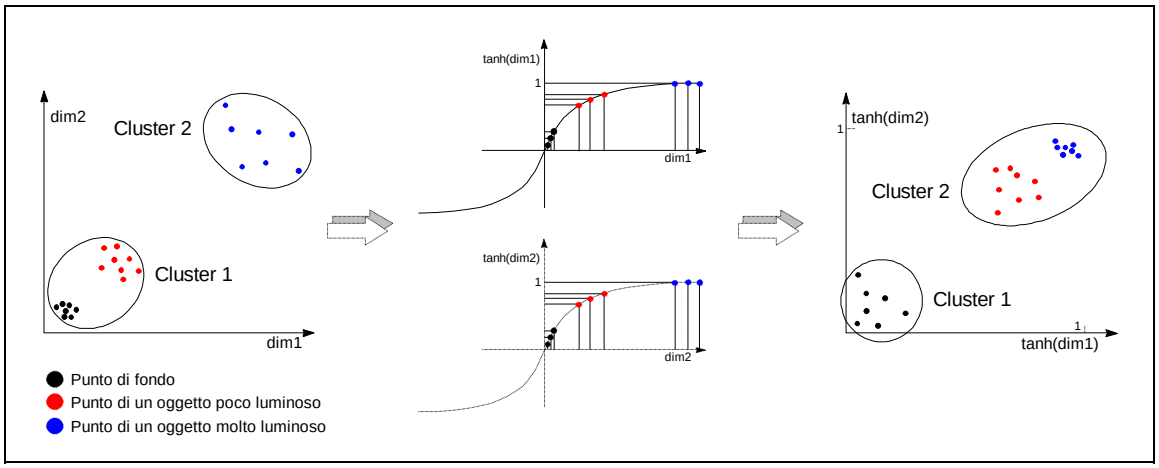

Fig. 6.6 - effetto dell'applicazione della tangente iperbolica sulla distribuzione di una PCA.

La nostra scelta è quindi caduta sulla rete *simmetrica robusta di generalizzazione della massimizzazione della varianza* (rete n. 7) che tra tutte quelle del *secondo tipo* è la più veloce (vedi la tabella di figura 6.3). Per quanto riguarda il numero di *componenti principali* selezionate, abbiamo constatato che, per qualsiasi algoritmo e per qualsiasi numero di lastre (nell'intervallo da 1 a 3), quasi il 90% dell'informazione è contenuto nelle prime tre componenti. Abbiamo quindi scelto di estrarre, per ogni *pattern*, solo queste prime tre componenti principali. Ciò comporta una riduzione di complessità nel funzionamento delle reti neurali dell'88% nel caso di segmentazione monobanda e del 97% nel caso di segmentazione multibanda rispetto all'utilizzo del *pattern* non ridotto.

6.4 Esperimenti di segmentazione.

E' necessario, a questo punto, stabilire con quale rete neurale non suprevisionata andremo a partizionare le proiezioni dei *pattern* ottenute dalla PCA in modo da identificarne le classi di appartenenza. Testeremo in questo paragrafo tutte le reti presentate nel capitolo 4 con particolare attenzione per quelle che ci permettono di evitare la fase di labelling. Sceglieremo, tra tutte, la rete che ci darà le segmentazioni migliori e, a parità di prestazioni, la più veloce.

Una prima incognita è il numero di classi che la rete deve individuare. Fino a questo momento abbiamo supposto che la rete debba discernere tra due sole classi: fondo e oggetto ma, come mostra la figura 6.7b, due classi non sono quasi mai sufficienti. Se si esaminano con attenzione le figure 6.4 e 6.5, si può notare che, pur applicando una PCA del *secondo tipo*, non si riescono a generare *cluster* spazialmente ben separati (abbiamo visto solo distribuzioni bidimensionali, ma per 3 dimensioni il discorso è analogo). Per poter ben segmentare con queste condizioni avverse, è necessario utilizzare un maggior numero di classi. Abbiamo stabilito dopo varie prove (si veda a questo scopo la figura 6.7 dove ogni colore rappresenta una classe) che il numero ottimale di classi è 6. Scegliere meno di 6 classi comporta un impoverimento della

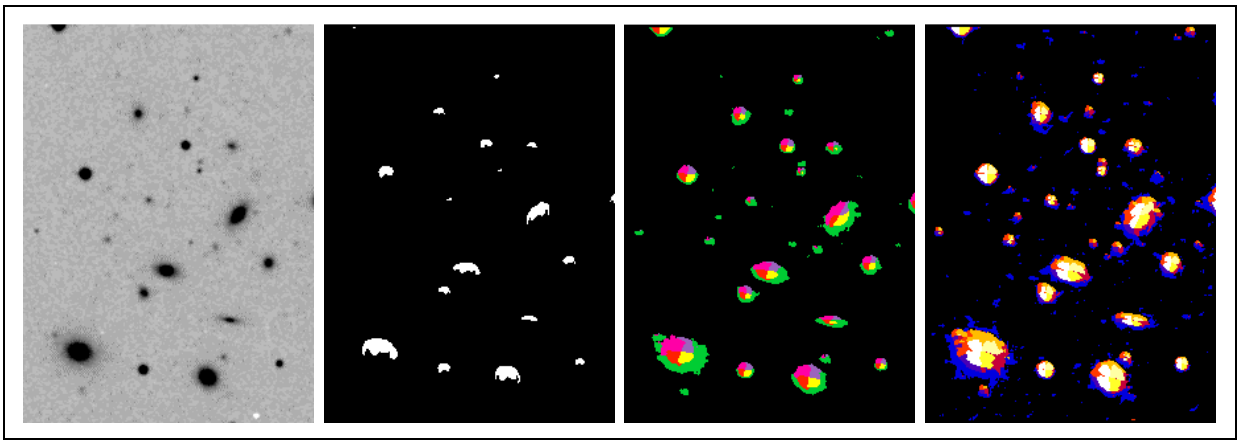

Fig. 6.7 - segmentazioni dell'immagine a sinistra con 2, 6 e 10 classi.

segmentazione, scegliere più di 6 classi comporta la generazione di segmentazioni rumorose.

Tra le classi identificate, solo una rappresenta il fondo e precisamente quella che comprende i *pattern* che rappresentano i pixel a più bassa luminosità. Tutte le altre rappresentano pixel appartenenti a diverse zone di oggetti. Si noti che le zone rappresentate da classi diverse non sono concentriche dato che l'elemento caratterizzante di ogni pixel non è la sua intensità ma il contesto in cui si colloca. E' frequente la divisione degli oggetti in *spicchi* come vedremo nelle prove di segmentazione delle figure 6.10, 6.11 e 6.12.

Passiamo ora ad analizzare le prestazioni delle reti neurali non supervisionate. In figura 6.8 sono mostrati i risultati ottenuti su un *training set* di 10000 *pattern*. Come c'era da aspettarsi, la distorsione media più bassa è raggiunta dalla *ML-NeuralGas* e dalle reti ibride (7, 8 e 9) immediatamente seguite dalla rete 5 (composta da un primo strato GCS e da due successivi strati NeuralGas). Le altre reti (a parte la ML-SOM) danno risultati piuttosto scadenti. Nulla però si può dire circa l'effettiva utilizzabilità delle reti, prima di aver esaminato le segmentazioni ottenute. A questo proposito, la figura 6.9 mostra diverse maschere ottenute dalla combinazione di una PCA di *secondo tipo* (rete n. 7) con ciascuna delle reti della tabella 6.8.

I risultati delle ***reti gerarchiche*** confermano quanto mostrato nella tabella dato che la rete migliore sembra essere proprio la *ML-Neural Gas*. Risultati molto simili si

	Tipo di rete	neuroni	distorsione	tempo (sec.)
1	ML-SOM	50 20 6	0.1188	340
2	ML-Neural Gas	50 20 6	0.1163	302
3	ML-K Means	50 20 6	0.1300	112
4	ML-Maximum Entropy	50 20 6	0.3478	217
5	GCS + ML-Neural Gas	50 20 6	0.1166	235
6	Neural Gas	6	0.2831	103
7	" " + Nearest Neighbour	50	0.1163	256
8	" " + met. del Centroide	50	0.1163	249
9	" " + media tra gruppi	50	0.1163	243

Fig. 6.8 - prestazioni delle reti neurali non supervisionate.

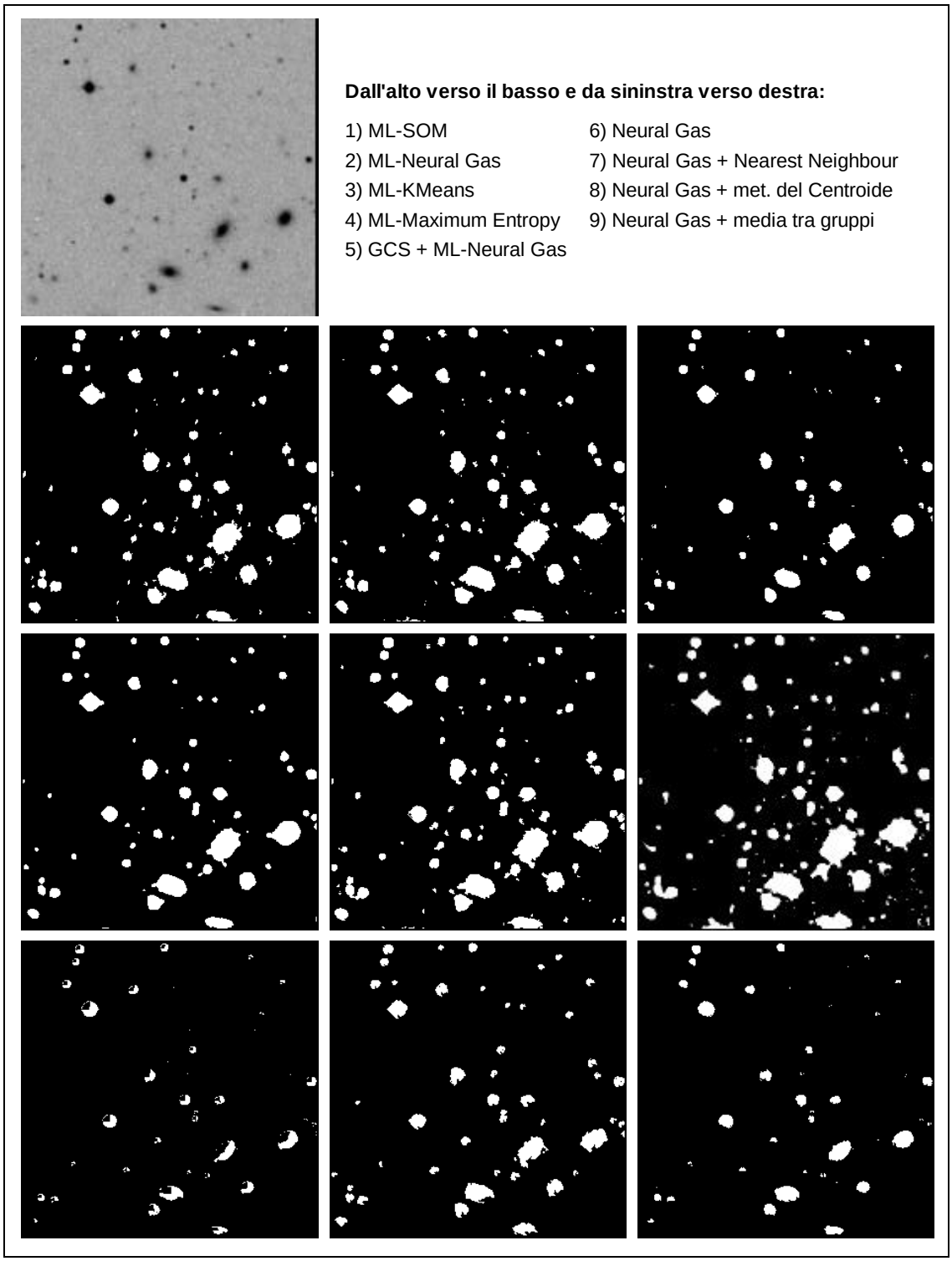

Fig. 6.9 - esperimenti di segmentazione dell'immagine in alto.

ottengono anche con la *ML-SOM* e con la rete 5 (*GCS + ML-Neural Gas*).

Per quanto riguarda le altre reti, si riscontrano situazioni opposte ma ugualmente negative: sia la rete *ML-KMeans* che la *ML-Maximum Entropy* perdono diversi oggetti confondendoli con il fondo; la *Neural Gas* con 6 neuroni invece, pur non perdendo nessun oggetto, aggiunge alla maschera parecchio rumore. Entrambi questi errori sono

	Tipo di rete	oggtti identificati	corretti
1	ML-SOM	757	699
2	ML-Neural Gas	771	720
3	ML-K Means	426	394
4	ML-Maximum Entropy	637	607
5	GCS + ML-Neural Gas	768	719
6	Neural Gas	1113	720
7	" " + Nearest Neighbour	384	354
8	" " + met. del Centroide	438	381
9	" " + media tra gruppi	308	297

Fig. 6.10 - prestazioni delle reti neurali non supervisionate.

dovuti alla difficoltà che hanno queste reti nel rappresentare correttamente il *training set* essendo quest'ultimo caratterizzato da un alto grado di *overlapping*.

I risultati delle ***reti ibride*** (7, 8 e 9) sono invece piuttosto deludenti ed in netto contrasto con il coefficiente di distorsione raggiunto. Il problema infatti non risiede nella rappresentazione del *training set* (utilizzando una *NeuralGas* di base i risultati sono, in questo senso, ottimi) ma nel raggruppamento dei 50 neuroni dello strato di output in 6 classi. L'algoritmo *nearest neighbour* tende ad inserire nel *cluster* di fondo anche neuroni che rappresentano oggetti molto deboli. Questa scelta determina un *effetto catena* che conduce alla perdita di parti di oggetti anche piuttosto luminosi. Purtroppo questo metodo funziona bene solo quando i cluster sono ben separati o quando il grado di overlapping è piuttosto basso. Risultati migliori, ma comunque inferiori a quelli della *ML-Neural-Gas*, si ottengono con i metodi del *centroide* e della *media tra gruppi*.

Le reti migliori restano quindi le reti 1, 2 e 5. Tra queste scegliamo senza dubbio la 5 che, unendo in un solo modello la precisione dell'algoritmo *Neural-Gas* con la velocità del *Growing-Cell*, riesce ad ottenere le prestazioni di una *ML-Neural Gas* sprecando solo il 78% di tempo. Una giustificazione quantitativa di questa affermazione si è ottenuta dando in input alla fase di valutazione degli oggetti (fase che descriveremo nel prossimo capitolo) una piccola immagine-campione segmentata tramite le diverse reti non supervisionate. I risultati di questo esperimento (mostrati in figura 6.10) confermano la nostra ipotesi basata inizialmente solo su una valutazione qualitativa delle maschere ottenute.

Per concludere mostriamo alcuni esperimenti di segmentazione su una banda (figure 6.11 e 6.12) e su tre bande (figura 6.13) ottenute con una PCA di tipo 7 e con una rete non supervisionata di tipo 5 (le immagini sono state invertite allo scopo di permetterne una migliore resa su carta). Non disponendo di maschere generate da SKICAT, non possiamo ancora fare alcun confronto. I confronti vengono quindi rimandati alle fasi successive.

Riferimenti.

N. Pal, S. Pal "A Review on Image Segmentation Techniques" in *Pattern Recognition*, vol. 26, no. 9, pp. 1277-1294, 1993.

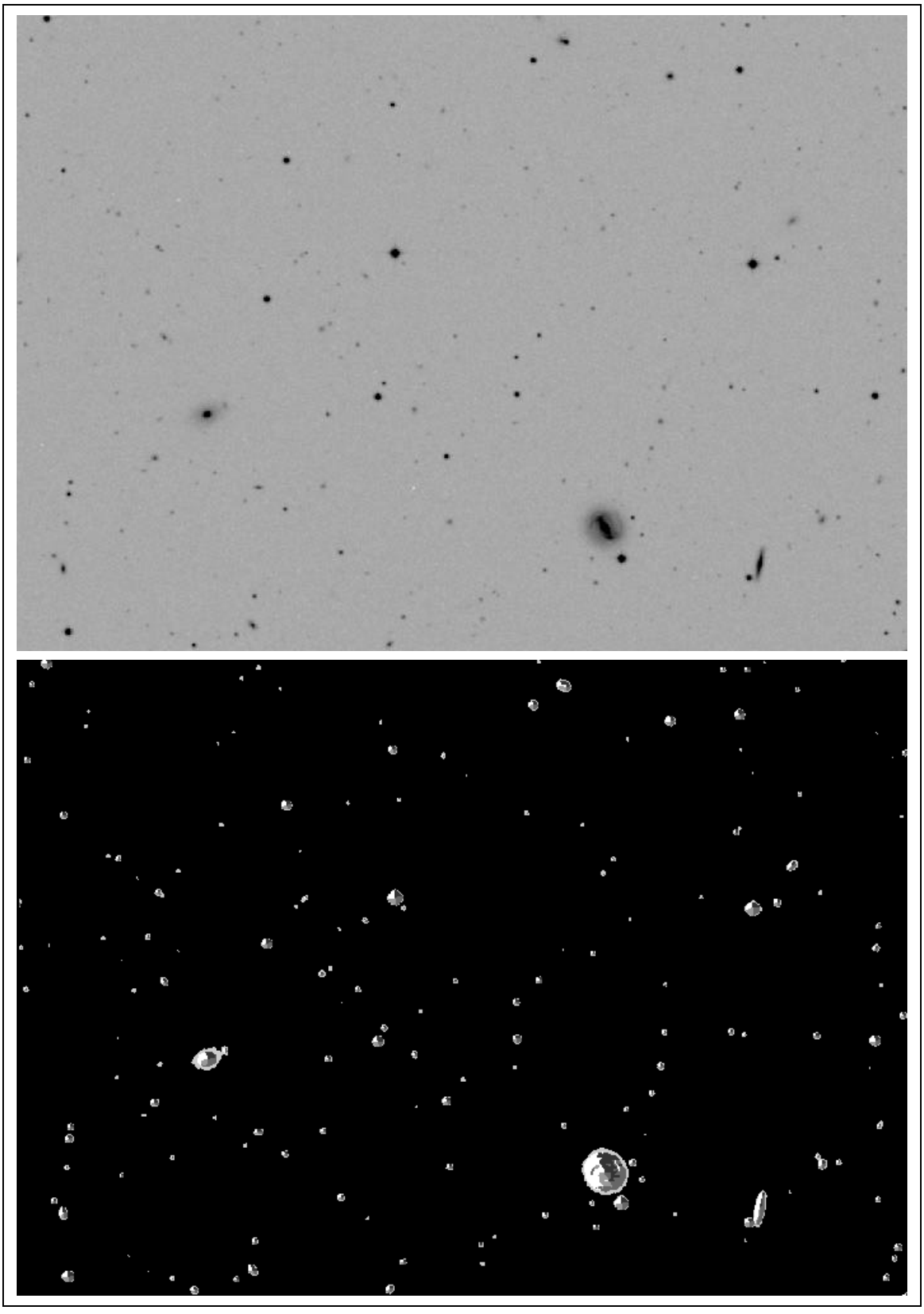

Fig. 6.11 - esempio di segmentazione su una banda.

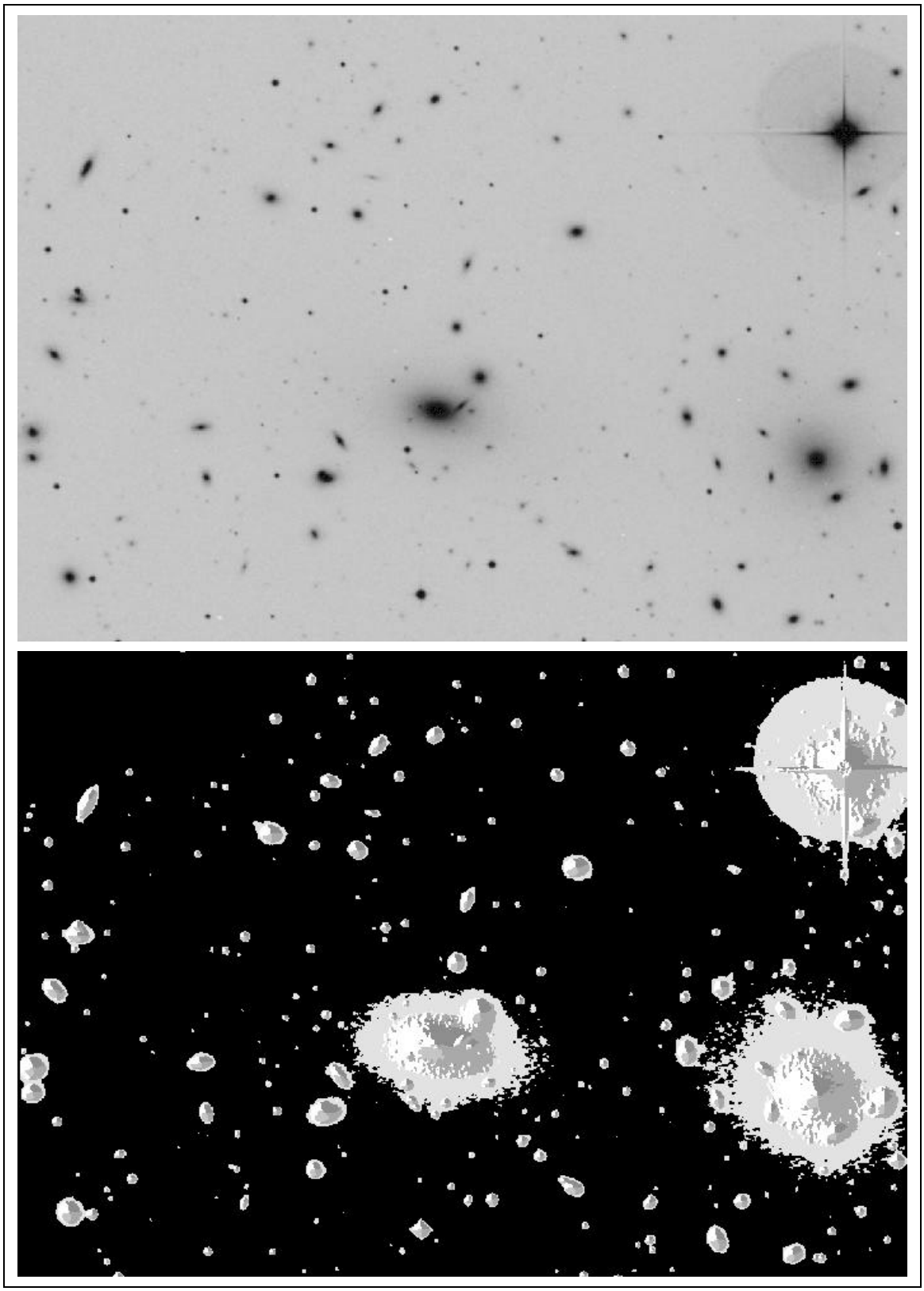

Fig. 6.12 - esempio di segmentazione in presenza di oggetti molto luminosi.

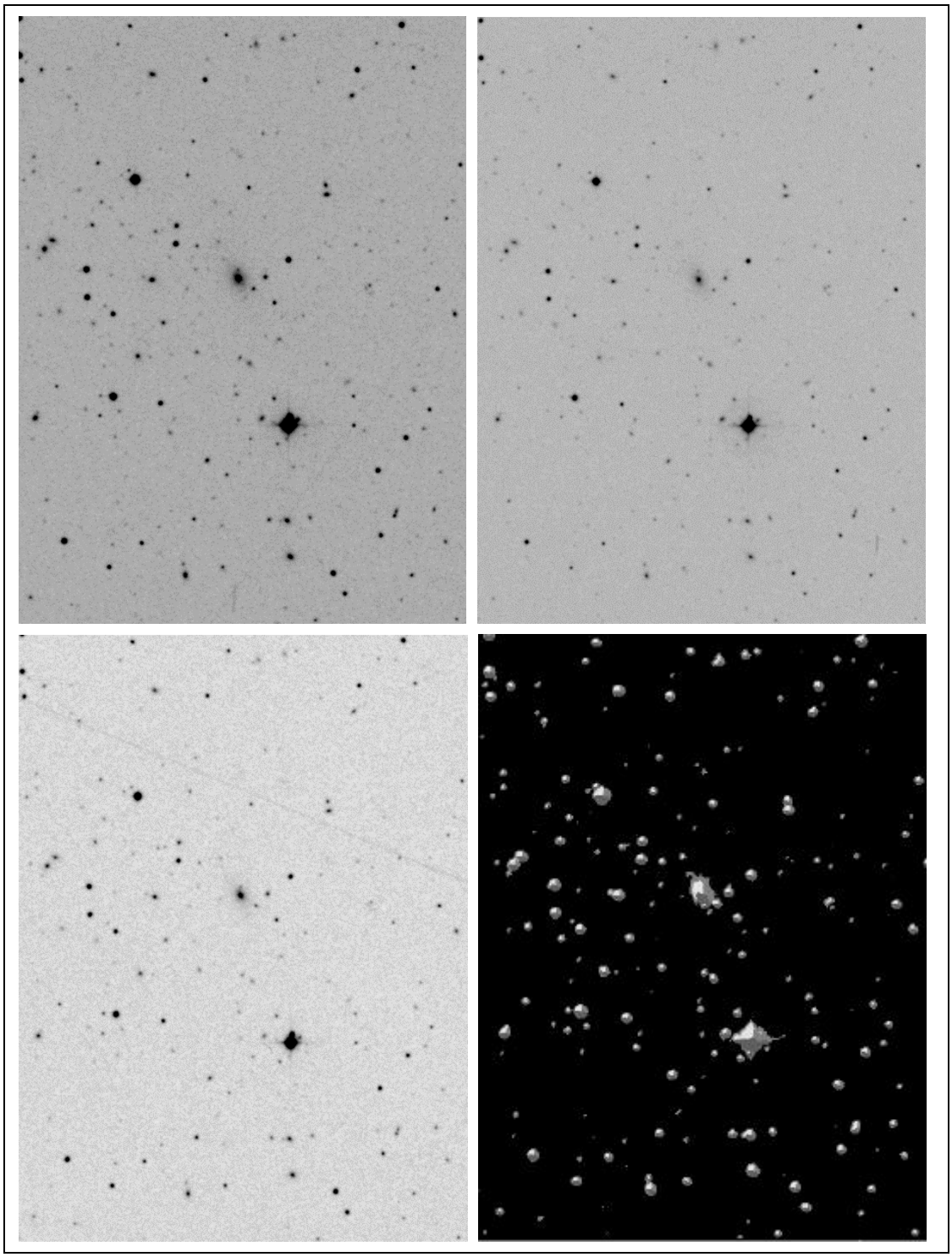

Fig. 6.13 - esempio di segmentazione su tre bande.

CAPITOLO 7

Secondo Passo: Valutazione degli Oggetti.

7.1 Introduzione.

Se nella fase di *segmentazione* si andavano ad identificare tutti gli oggetti contenuti in una lastra, nella fase di *valutazione* si deve misurare, per ognuno di questi oggetti, una serie di parametri fotometrici da usare come 'input per il *classificatore*. L'input della fase di *valutazione* è costituito, oltre che dalla lastra da processare, anche dalla *maschera* ottenuta nella fase precedente. L'output sarà invece un file dove ogni record rappresenta un oggetto rilevato sulla lastra e, ogni campo, una particolare caratteristica estratta.

La prima cosa da fare è ***rilevare gli oggetti*** dalla maschera ed inserirli in apposite strutture. Ogni blocco di pixel attivati adiacenti è un oggetto e può essere rilevato con un semplice algoritmo che scandisce tutta la maschera pixel per pixel dall'alto verso il basso e da sinistra verso destra. Questo algoritmo, per ogni pixel attivato, andrà a considerare tutti gli oggetti ad esso adiacenti. Si potranno verificare tre casi:

- se il pixel non tocca nessun oggetto allora viene inizializzato un nuovo oggetto che conterrà quel solo pixel;
- se il pixel tocca un solo oggetto allora il pixel viene aggiunto ad esso;
- se il pixel tocca più oggetti allora questi vengono fusi tra loro a costituire un unico oggetto e il pixel viene aggiunto ad esso.

Lo stesso algoritmo può essere velocizzato considerando non un singolo pixel alla volta ma una sequenza orizzontale di pixel attivati. La figura 7.1 mostra quattro iterazioni successive di rilevamento. C'è inizialmente (figura 7.1a) un oggetto completo (rosso) e uno in fase di costruzione (blu). Si processa una nuova linea che, non toccando nessun oggetto già esistente, viene inserita in un nuovo oggetto (verde); si processa la linea successiva (figura 7.1b) che, toccando un solo oggetto (blu), viene aggiunta ad esso (figura 7.1c); si processa infine una nuova linea che, toccando due oggetti diversi, viene inserita nella loro unione (figura 7.1d).

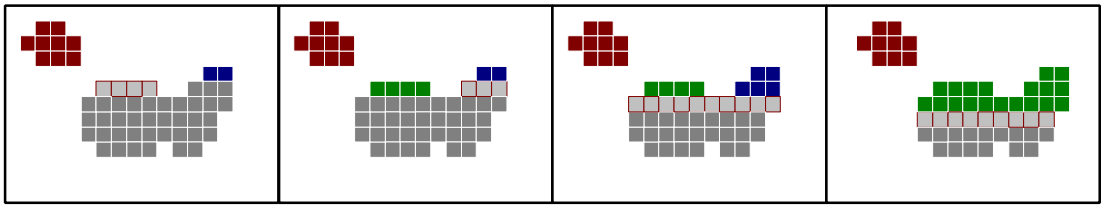

Fig. 7.1 - quattro iterazioni successive di rilevamento di oggetti sulla maschera.

Una volta rilevati in questo modo tutti gli oggetti della lastra, si potrebbe pensare di passare direttamente all'estrazione delle caratteristiche. Purtroppo la cosa non è così semplice dato che, in genere, gli oggetti ottenuti dalla *segmetazione* non sono una buona rappresentazione dei corpi celesti di cui sono l'immagine.

Un primo problema che si può presentare (e che si presenta di frequente) è che un oggetto è ***multiplo*** ovvero che rappresenta non uno ma più corpi celesti distinti. Gli oggetti di questo tipo si generano quando più corpi celesti si trovano talmente vicini (sulla lastra ma non necessariamente nello spazio) da apparire non risolti (le loro dimensioni angolari sono confrontabili o lievemente maggiori della loro separazione angolare). L'algoritmo di *segmentazione* identifica erroneamente l'intero blocco come se fosse un singolo oggetto. Per ovviare a questo difetto abbiamo realizzato un algoritmo di separazione intelligente che descriveremo nel prossimo paragrafo.

Un'altra operazione che risulta necessaria prima di poter procedere all'estrazione delle caratteristiche, è l'*affinamento dei contorni* degli oggetti rilevati dato che la *segmentazione* crea spesso forme non perfettamente rispondenti a quelle dei corpi celesti che vuole rappresentare. In fase di affinamento dobbiamo attenuare questo sfasamento cercando di far corrispondere i centri e di coprire tutta l'area informativa. Solo in questo modo l'*estrazione* sarà efficace. Il nostro algoritmo di affinamento è mostrato nel paragrafo 7.3.

Nel paragrafo 7.4 parleremo infine di come è stata implementata l'*estrazione delle caratteristiche* e ci soffermeremo sulla descrizione dei parametri scelti. Abbiamo

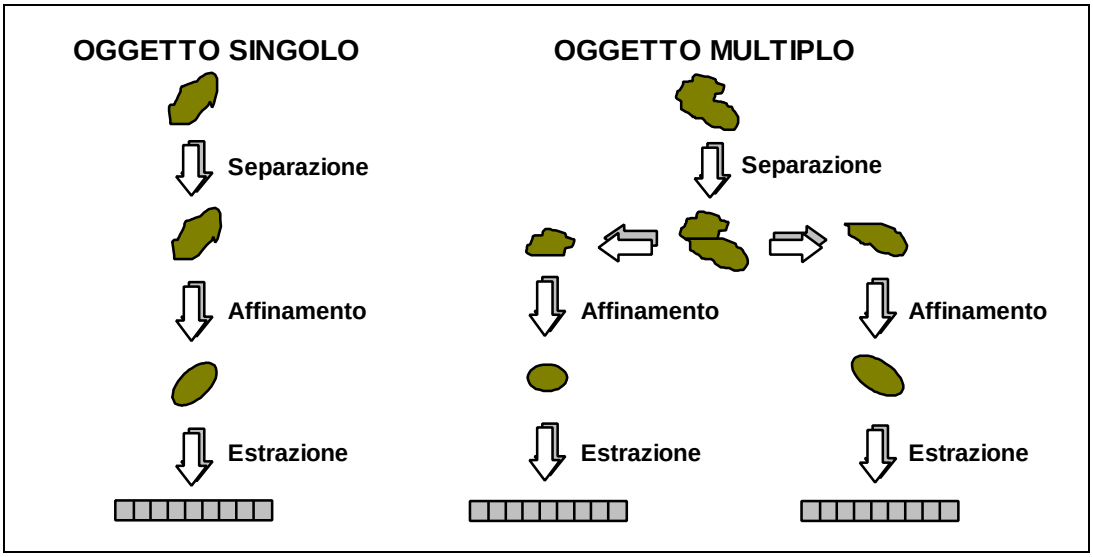

Fig. 7.2 - schema di funzionamento della valutazione degli oggetti.

preferito estrarre molti parametri anche se, come vedremo nel prossimo capitolo, non tutti sono stati utilizzati per la classificazione. Nell'ultimo paragrafo mostreremo infine alcuni esperimenti di estrazione di caratteristiche su immagini reali e confronteremo i nostri risultati con quelli ottenuti da SKICAT; il nostro approccio, come vedremo, garantisce prestazioni di gran lunga superiori. La figura 7.2 sintetizza il processing effettuato sugli oggetti nella fase di valutazione.

7.2 Separazione degli oggetti multipli.

Come abbiamo già accennato, se due corpi celesti presumibilmente distinti sono tanto vicini sulla latra da sovrapporsi, la *segmentazione* tende ad identificarli come un singolo blocco di pixel. Ciò comporterebbe l'estrazione di false caratteristiche e si risolverebbe in una errata classificazione: un problema che diviene tanto più grave quanto più affollati sono i campi in esame (ad esempio immagini di regioni a bassa latitudine galattica). Bisogna quindi trovare il modo di separare il blocco e ripristinare una corrispondenza più fedele tra oggetti rilevati e corpi celesti presenti sulla lastra: ad ogni oggetto deve corrispondere uno e un solo corpo celeste. Descriveremo in questo paragrafo la nostra soluzione.

Dato che la stragrande maggioranza dei corpi celesti ha forma circolare o ellittica, possiamo pensare di rappresentare un qualsiasi oggetto semplicemente conservando i parametri dell'ellisse che lo circoscrive. La prima cosa da fare è, quindi, individuare il ***centro*** $(\bar{x},\bar{y})$ dell'oggetto in questo modo:

$$\bar{x}=\frac{1}{M_{00}}\sum_{(x,y)\in A} x\cdot D(x,y) \quad \text{e} \quad \bar{y}=\frac{1}{M_{00}}\sum_{(x,y)\in A} y\cdot D(x,y) \tag{7.1}$$

dove A è l'insieme che rappresenta l'area dell'oggetto e ne contiene tutte le coordinate dei punti; $D(x,y)$ è la densità rilevata nel punto (x,y) sulla lastra (il valore del pixel (x,y)) mentre M_{00} è il momento zero dell'oggetto definito come:

$$M_{00}=\sum_{(x,y)\in A} D(x,y). \tag{7.2}$$

Un altro parametro da determinare è la lunghezza del ***semiasse maggiore*** che può essere ottenuta come la più grande distanza tra i punti di A e il centro $(\bar{x},\bar{y})$ ovvero:

$$a=\max_{(x,y)\in A} \left\|(x,y)-(\bar{x},\bar{y})\right\|. \tag{7.3}$$

Per velocizzare il procedimento possiamo considerare solo i punti del contorno di A dato che il punto più distante dal centro si troverà sicuramente sul bordo. Un semplice calcolo trigonometrico di consente di stabilire anche l'***angolo*** di incidenza del prolungamento del semiasse maggiore sull'asse delle ascisse (noto in astronomia come Angolo di Posizione del Semiasse maggiore fotometrico o APS) dato da:

$$a=\operatorname{arctg}\left(\frac{y'-\bar{y}}{x'-\bar{x}}\right) \tag{7.4}$$

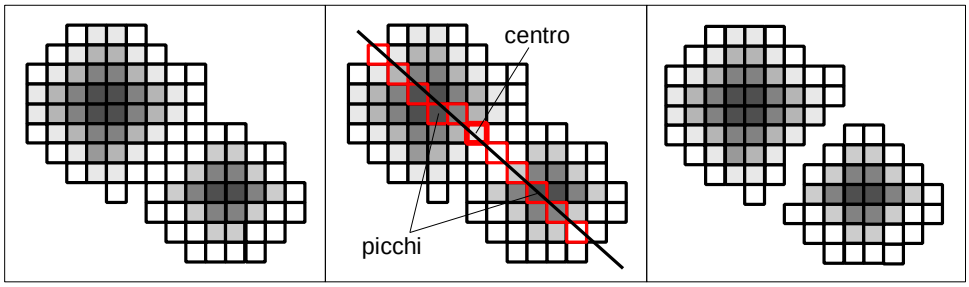

Fig. 7.3 - esempio di separazione di un oggetto multiplo.

dove la coppia (x', y') rappresenta le coordinate (ottenute al passo precedente) del punto dell'oggetto più distante dal centro. Andiamo infine a calcolare la lunghezza del ***semiasse minore*** (anche se questo parametro non ci interessa ai fini della separazione) come distanza più grande tra i punti di A e la retta del semiasse minore in questo modo:

$$b = \max_{(x,y)\in A} \left| \text{sen}\left[\text{arctg}\left(\frac{y-\bar{y}}{x-\bar{x}} \right) - a \right] \right| \cdot \left\| (x,y) - (\bar{x},\bar{y}) \right\| \tag{7.5}$$

dove, anche in questo caso, potremmo limitarci a considerare solo i punti del bordo.

Una volta ottenuti questi 4 parametri possiamo procedere all'identificazione degli ***oggetti multipli***. Il principio è semplice: si effettua uno *scanning* dei pixel dell'oggetto lungo un asse che passa attraverso il centro $(\bar{x},\bar{y})$ a un angolo b_i al fine di riconoscere un doppio picco di luminosità. Se si riscontrano due picchi (due punti di massimo) infatti, si assume che l'oggetto sia *multiplo* e si procede alla ***separazione*** tracciando una retta perpendicolare all'asse e passante per il punto a luminosità più bassa tra i due picchi. La figura 7.3 mostra schematicamente questo procedimento (i pixel bordati in rosso sono quelli dell'asse ispezionato). Ogni oggetto risultante dall'operazione di separazione viene riprocessato e ritestato per la presenza di doppi picchi. Per quanto riguarda l'angolo b_i, questo viene settato inizialmente pari ad dato che, se un doppio picco ci deve essere, questo è più probabile che si presenti lungo l'asse maggiore che è la linea ad intersezione più grande con l'oggetto. Se non si trovano doppi picchi lungo quest'asse si tenta con altri angoli in modo da ispezionare un massimo di n assi equidistanti. Gli angoli b_i possono quindi essere così ottenuti:

$$b_i = a + ip/n \quad \text{per} \quad 0 \le i < n. \tag{7.6}$$

Fino a questo punto, l'algoritmo è simile a quello implementato dal FOCAS a parte il fatto che quest'ultimo si limita a ispezionare solo i punti dell'asse maggiore. Ispezionare anche altri assi ci porta ad ottenere ottime separazioni anche in presenza di oggetti ad alta ellitticità con una spesa computazionale irrisoria.

Una pecca di questo algoritmo è che si possono verificare circostanze in cui un oggetto singolo (non multiplo) con anomalie di luminosità dovute a del rumore viene erroneamente separato. La figura 7.4a mostra un caso in cui la presenza di pixel rumorosi su uno degli assi ispezionati genera un doppio picco di luminosità e quindi attiva la procedura di separazione.

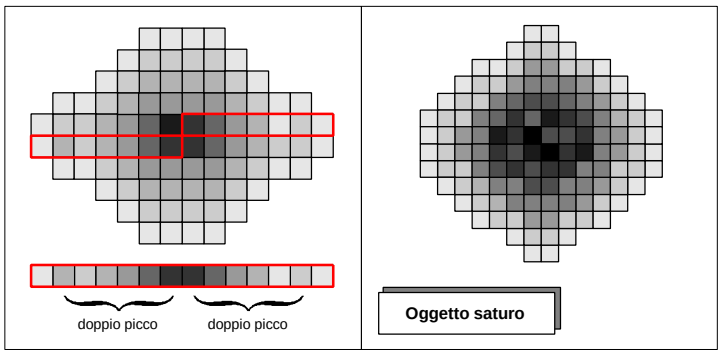

Fig. 7.4 - oggetti con anomalie di luminosità.

Purtroppo le anomalie di luminosità sono piuttosto frequenti. Esistono, inoltre, corpi celesti con luminosità al centro talmente alta da superare il limite di saturazione di una lastra fotografica. Il risultato è che il nucleo dell'oggetto verrà rilevato (in maniera erronea) come una distribuzione caotica di luminosità. Un oggetto di questo tipo è detto *saturo* (figura 7.4b) ed è una vera dannazione per un algoritmo del genere dato che dà luogo ad una miriade di picchi in una zona (il nucleo) altamente informativa. Gli oggetti *saturi*, come vedremo nel paragrafo 7.5, sono malamente trattati dal FOCAS che li frantuma in una grande quantità di oggetti spuri.

Prima di vedere come si può migliorare l'algoritmo, presentiamo ancora un altro problema (vedi figura 7.5) che si verifica quando l'oggetto multiplo è composto da tre o più corpi celesti. Dato che la separazione si basa sulla ricerca di un doppio picco, essa coinvolge al più due corpi celesti. Ciò implica che la linea di separazione dividerà questi due corpi non tenendo conto della presenza di altri corpi nello stesso oggetto che potranno essere spezzati irreparabilmente in più parti. In figura 7.5a, una prima separazione dà luogo ai due oggetti di figura 7.5b che, successivamente, vengono separati nei quattro di figura 7.5c. Il corpo B viene quindi spezzato in due.

Un primo approccio per attutire questi inconvenienti è stato quello di ridurre il *range* dei valori di luminosità per i pixel degli oggetti. Le informazioni che si hanno sulla luminosità sono infatti troppo dettagliate: in questa fase ci interessa solamente una informazione generale sulla sua tendenza al crescere o al decrescere. Tenendo conto di queste necessità, bisogna ridefinire il valore di ogni pixel per farlo cadere entro un intervallo discreto di ampiezza *N*. Nel fare questa trasformazione si deve anche tener conto del valore di saturazione *Sat* della lastra come valore al di sopra del quale si hanno distribuzioni di luminosità caotiche (questo valore viene in genere fornito con la lastra). Si può così calcolare la luminosità *modificata* $D_N(x,y)$ per ogni

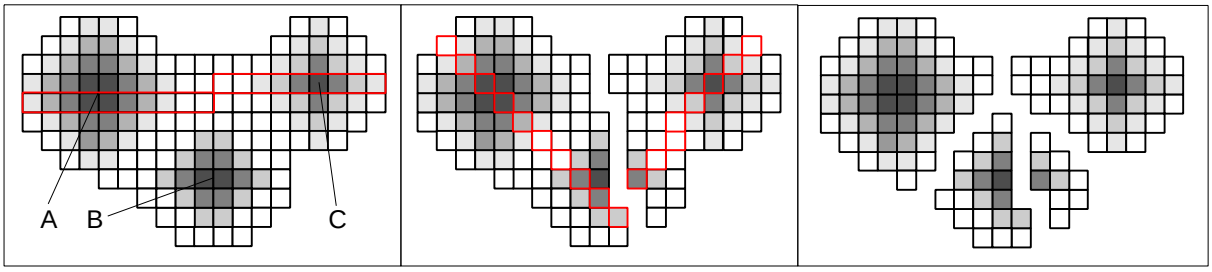

Fig. 7.5 - esempio di separazione errata: un corpo viene spezzato in due parti.

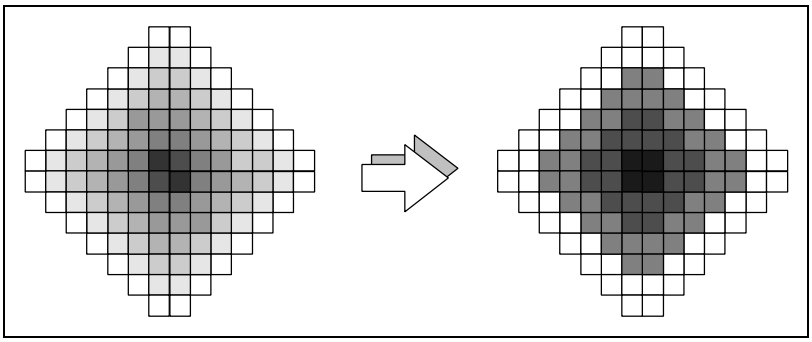

Fig. 7.6 - effetti della riduzione del range di luminosità.

pixel $(x,y) \in A$ per ogni oggetto A della lastra in questo modo:

$$D_N(x,y) = \begin{cases} N & \text{se } D(x,y) > Sat \\ \left[N \cdot (D(x,y) - D^A_{\min}) / (Sat - D^A_{\min}) \right] & \text{se } D(x,y) \le Sat \text{ e } D^A_{\max} > Sat \\ \left[N \cdot (D(x,y) - D^A_{\min}) / (D^A_{\max} - D^A_{\min}) \right] & \text{se } D(x,y) \le Sat \text{ e } D^A_{\max} \le Sat \end{cases} \qquad (7.7)$$

dove $D^A_{\min}$ e $D^A_{\max}$ sono, rispettivamente, i valori minimo e massimo delle luminosità dei pixel di A mentre la funzione $[x]$ individua l'intero più vicino ad x. Utilizzando questo metodo, come si può notare dall'esempio mostrato in figura 7.6, si riducono gli inconvenienti dovuti alle piccole anomalie di luminosità ed alla presenza di oggetti saturi. L'*oversplitting* degli oggetti (la loro tendenza ad essere separati più del dovuto) viene quindi attenuato ma non del tutto risolto. Un successivo miglioramento si può ottenere non modificando ulteriormente l'algoritmo di separazione ma aggiungendo una fase di ***ricomposizione degli oggetti*** erroneamente separati.

Dato che i corpi celesti sono in genere costituiti da un nucleo molto luminoso che degrada via via verso l'esterno, se un oggetto è stato erroneamente separato in due parti, tra i punti che toccano entrambi i suboggetti ottenuti dalla separazione, ce ne sarà almeno uno con valore di luminosità molto alto (si veda il corpo B in figura 7.5). D'altra parte, se un oggetto è stato ben separato, tutti i punti che toccano entrambi i suboggetti avranno valori di luminosità bassi dato che sono tutti punti di fondo (si vedano i corpi A e C in figura 7.5). Se indichiamo con A l'oggetto multiplo e con S_A l'insieme di suboggetti ottenuti da A dall'algoritmo di separazione, possiamo pensare

1. Si ordini l'insieme $S_A = \{A_1, A_2, \ldots, A_m\}$ in ordine decrescente rispetto alla luminosità media $D^{A_i}_{\text{med}}$; si ponga inoltre p=1 e q=p+1.

2. Si considerino i suboggetti A_p e A_q: se esiste almeno un punto (x,y) adiacente ad entrambi e tale che: $D(x,y) > D^{A_q}_{\text{med}}$ allora si unisca ad A_p il suboggetto A_q e tutti i punti intermedi tra i due.

3. Se al punto **2** è avvenuta una ricomposizione allora si elimini A_q dall'insieme S_A, lo si rinumeri e si ponga q=p+1; in caso contrario si incrementi q. Se $q \le m$ si torni al punto **2**.

4. Se p<m-1 allora si incrementi p, si ponga q=p+1 e si torni al punto **2**; altrimenti **STOP**.

Fig. 7.7 - algoritmo di ricomposizione: prima versione.

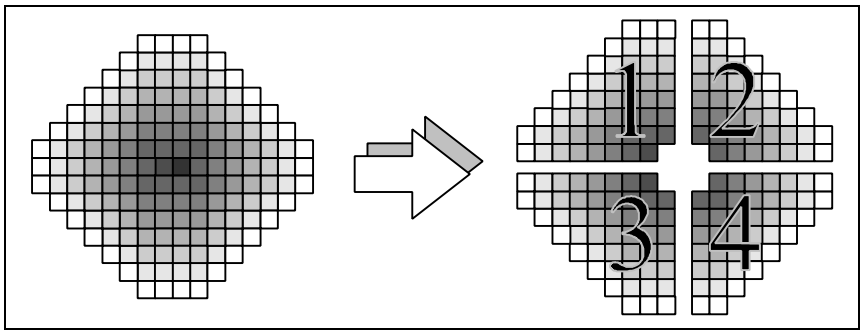

Fig. 7.8 - separazione di un oggetto in quattro settori.

ad un primo algoritmo di ricomposizione come mostrato in figura 7.7.

L'algoritmo prende in esame un suboggetto alla volta partendo da quello a luminosità media più alta e lo confronta con tutti gli altri per eventuali ricomposizioni. Si procede ad una ricomposizione quando tra due di questi suboggetti esistono punti con un valore di luminosità maggiore della più piccola luminosità media dei due suboggetti. In questo caso infatti (vedi ancora la figura 7.5) è plausibile pensare che verso i punti individuati si abbia una luminosità crescente e quindi si può concludere che gli oggetti debbano essere uniti. Viceversa, se non si ha nessun valore maggiore della media, i punti adiacenti ai due oggetti individuano una luminosità decrescente e quindi appartengono a zone periferiche di corpi celesti separati. In questo caso la ricomposizione non deve avvenire. Considerando i suboggetti in ordine decrescente di luminosità media riusciamo ad agganciare agli oggetti più luminosi tutti i suboggetti formati dai frammenti del suo alone. Prendendo in esame gli oggetti in un qualsiasi altro ordine potremmo erroneamente associare ad oggetti poco luminosi gli aloni generati da altri corpi.

Quando un corpo celeste debole *A* si trova vicino ad un corpo molto luminoso *L* esso subisce un'influenza che si traduce in un'interferenza nella sua funzione di luminosità. Se i due corpi sono tanto vicini da essere uniti in un solo oggetto dalla segmentazione, questi verranno certamente disgiunti dall'algoritmo di separazione. Il problema è che rischiano di essere nuovamente uniti dall'algoritmo di ricomposizione. *L* tende infatti ad influenzare (aumentandola) la luminosità dei punti di *A* ad esso più vicini compresi i punti di separazione tra i due suboggetti (l'influenza scema con l'allontanarsi da *L*). A volte capita che, pur essendo *A* ed *L* corpi separati, esistono punti intermedi con luminosità superiore alla luminosità media di *A*: in questo caso la ricomposizione è immediata.

Per evitare questa occorrenza si può pensare di dividere *A* in *n spicchi* (settori) come mostra la figura 7.8, calcolare la luminosità media $D_{\text{med}}^{A_k}$ per ciascuno di essi e confrontare le luminosità di ciascun pixel intermedio con le luminosità medie dei soli spicchi di *A* ad esso adiacenti piuttosto che con quella globale. In questo modo, i settori di *A* più vicini ad *L* saranno sicuramente più luminosi (in media) dei punti intermedi (ciò è dovuto in parte all'influenza di *L*) e quindi gli oggetti non verranno uniti dato che non intervengono ad abbassare la media i settori di *A* più lontani da *L* (quindi meno luminosi perché non influenzati). Apportando queste modifiche al

Si consideri l'algoritmo in **figura 7.7** e si sostituisca il solo punto **2** con il seguente:

2. Si considerino i suboggetti A_p e A_q: se esiste almeno un punto (x, y) adiacente ad entrambi e tale che:

$$D(x,y) > \max_{k \text{ adiacente a } (x,y)} D_{\text{med}}^{A_{qk}}$$

allora si unisca ad A_p il suboggetto A_q e tutti i punti intermedi tra i due.

Fig. 7.9 - algoritmo di ricomposizione: seconda versione.

precedente algoritmo (come mostrato in figura 7.9) otteniamo finalmente la versione adottata dal nostro programma.

7.3 Affinamento dei contorni.

Se la segmentazione genera spesso oggetti di forma non corrispondente a quella dei corpi che vuole rappresentare (come abbiamo già accennato nel paragrafo 7.1), c'è da dire che l'algoritmo di separazione da noi adottato peggiora sovente questa situazione. Per attenuare lo sfasamento tra i corpi celesti della lastra e gli oggetti utilizzati ai fini dell'estrazione delle caratteristiche, sono state adottate alcune tecniche che nel seguito descriveremo.

Un metodo efficace per ottenere la reale forma di un corpo *A* consiste nell'eliminare dall'oggetto che lo rappresenta tutti i pixel con luminosità inferiore ad una certa soglia (questa tecnica prende il nome di *tresholding*). Dopo il *tresholding*, l'oggetto sarà delimitato da un'*isofota* (insieme dei punti con la stessa brillanza) che, in condizioni normali, individua la forma esatta del corpo (vedi figura 7.10). In condizioni non normali, ovvero in prossimità di un corpo molto luminoso *L*, le isofote tendono ad allungarsi a causa dell'interferenza di *L* e, di conseguenza, non sono più buone rappresentazioni della forma. Tuttavia, man mano che ci si avvicina al centro di *A*, le isofote tendono a diventare sempre meno allungate perché la luminosità dovuta ad *A* diventa dominante rispetto all'influenza di *L*. Si può pensare, quindi, di effettuare su *A* un *tresholding* forte (con una soglia molto alta) per individuarne il nucleo (la cui forma

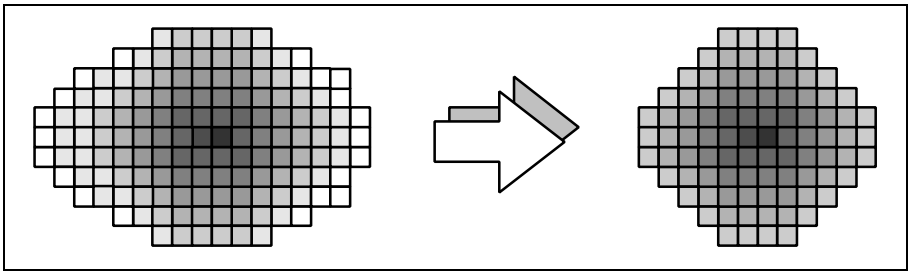

Fig. 7.10 - individuazione della forma tramite tresholding.

1. Si effettui sull'oggetto *A* un tresholding a soglie *s* successive e tali che:

$$D^A_{\min} \le s \le (D^A_{\max} - D^A_{\min})\alpha + D^A_{\min} \quad \text{dove} \quad 0 \le \alpha \le 0.5$$

finché $|A| \cong \pi ab$ (dove *a* e *b* sono calcolati tramite la 7.3 e la 7.5) o finché *s* non raggiunge il massimo valore consentito.

2. Si effettui su *A* un tresholding forte con $s = (D^A_{\max} - D^A_{\min})\beta + D^A_{\min}$ dove $0.5 \le \beta \le 1$ e si conservi il nuovo oggetto (nucleo) in *B*.

3. Si individuino i seguenti pixel: x_A come intersezione più lontana dal centro di *B* del prolungamento dell'asse maggiore di *B* con il bordo di *A*; x_B come intersezione più vicina ad x_A dell'asse di *B* con il bordo di *B* stesso.

4. Se $x_A = x_B$ allora si prosegua con il passo **5**. In caso contrario si individui il pixel x_C come punto del prolungamento dell'asse di *B* equidistante da x_A e x_B. Se $D(x_B) > D(x_C)$ si pone $x_B = x_C$ altrimenti si pone $x_A = x_C$. Si ritorni al passo **4**.

5. Si sostituisca l'oggetto *A* con l'oggetto centrato nel centro $(\bar{x}, \bar{y})$ di *B*, avente come semiasse maggiore il segmento $(\bar{x}, \bar{y})\, x_A$ e un semiasse minore di lunghezza pari a:

$$\overline{(\bar{x}, \bar{y})x_A} \cdot b/a$$

dove *a* e *b* sono le lunghezze dei semiassi maggiore e minore di *B*.

Fig. 7.11 - algoritmo di affinamento dei contorni.

è corretta per quanto abbiamo detto) e poi espandere l'oggetto così ottenuto mantenendo costanti le proporzioni tra gli assi.

In base a queste considerazioni nasce l'algoritmo di affinamento dei contorni descritto in figura 7.11 dove il ciclo del punto 4 assicura che l'espansione dell'oggetto si interrompa all'invertirsi della tendenza al decrescere della luminosità (calcolata a partire dal nucleo). In questo caso l'oggetto ha già inglobato completamente il corpo che vuole rappresentare e quindi se ne può arrestare la crescita.

7.4 Estrazione delle caratterstiche.

L'output della fase di valutazione, come abbiamo già accennato nel paragrafo 7.1, è un file di parametri calcolati sugli oggetti riconosciuti in fase di segmentazione. Ogni riga di questo file rappresenta un oggetto ed ogni colonna una particolare misurazione relativa all'oggetto stesso. Gli algoritmi dei due precedenti paragrafi erano tesi a rendere più accurata l'estrazione di questi parametri (caratteristiche) eliminando dagli oggetti tutte le impurità aggiunte in sede di segmentazione. Nel seguito daremo una breve descrizione dei 25 parametri che abbiamo scelto di estrarre (sono tra quelli più utilizzati in letteratura) ma si consiglia di consultare i riferimenti per una descrizione più dettagliata.

Le prime 5 caratteristiche descrivono l'ellisse che circosrive l'oggetto e sono: le ***coordinate del centro*** (ottenute con la 7.1), la ***lunghezza dei semiassi maggiore e***

minore (ottenute con la 7.3 e la 7.5) e l'***angolo*** di incidenza del prolungamento del semiasse maggiore sull'asse delle ascisse (ottenuto con la 7.4). Si tratta di parametri già calcolati in fase di *affinamento* e che possiamo riutilizzare. A questi si aggiunge un altro parametro $|A|$ che descrive l'***area*** dell'oggetto vista come numero di pixel che lo compone. Anche questo è un parametro di cui già disponiamo.

A queste, si aggiungono altre 12 caratteristiche suggerite da *Odewahn* nel 1992 (vedi i riferimenti alla fine del capitolo) e precisamente: il ***diametro*** dell'oggetto $dia = 2a$; l'***ellitticità*** $ell = 1 - b/a$; il ***valor medio di trasmissione*** (media delle densità dei pixel) $T_{av} = M_{00}/|A|$; il ***valore centrale di trasmissione*** (densità del pixel centrale) $T_c = D(\bar{x}, \bar{y})$; il ***rapporto tra le aree*** $c_1 = pa^2/|A|$ dove pa^2 è l'area del cerchio con raggio pari al semiasse maggiore dell'ellisse che circoscrive l'oggetto; il ***logaritmo dell'area*** $c_2 = \log(|A|)$; il ***momento*** M_1 ottenuto con la seguente formula:

$$M_1 = \sum_{(x,y)\in A} \frac{D(x,y)}{M_{00}||(x,y)-(\bar{x},\bar{y})||}; \tag{7.8}$$

più 5 semplici ***gradienti*** dell'immagine G_{14}, G_{13}, G_{12}, G_{23} e G_{34} ottenuti come segue:

$$G_{ij} = \frac{T_j - T_i}{r_i - r_j} \tag{7.9}$$

dove T_i è il valor medio di trasmissione su un anello ellittico centrato in $(\bar{x}, \bar{y})$, di angolo , di semiasse maggiore $r_i < a$ e di ellitticità *ell*. Si scelgono, a questo proposito, quattro raggi r_1, ... ,r_4 equidistanti l'uno dall'altro definendo $r_i = ia/4$.

Miller nel 1996 (vedi i riferimenti) propose di aggiungere alla lista di *Odewahn* altri due parametri facilmente ottenibili e altamente discriminanti e precisamente i due ***rapporti*** $T_r = T_{av}/T_c$ e $T_{cA} = T_c/\sqrt{|A|}$ che abbiamo provveduto a calcolare.

Abbiamo infine estratto le 5 caratteristiche del FOCAS ovvero: ***secondo e quarto momento totale*** ottenuti come:

$$C_2 = \frac{M_{20} + M_{02}}{M_{00}} \quad \text{e} \quad C_4 = \frac{M_{40} + 2M_{22} + M_{04}}{M_{00}}; \tag{7.10}$$

dove gli M_{ij} sono i momenti centrali dell'oggetto calcolati in questo modo:

$$M_{ij} = \sum_{(x,y)\in A} (x-\bar{x})^i (y-\bar{y})^j D(x,y); \tag{7.11}$$

lo ***schiacciamento*** dell'oggetto:

$$E = \frac{\sqrt{(M_{20} - M_{02})^2 + 4M_{11}^2}}{M_{02} + M_{20}}; \tag{7.12}$$

la ***densità di picco*** ovvero la densità media calcolata in un'area 33 centrata in $(\bar{x}, \bar{y})$ e, per concludere, il ***raggio efficace*** dell'oggetto definito come segue:

$$R_{ef} = \frac{1}{M_{00}} \sum_{(x,y)\in A} D(x,y)||(x,y)-(\bar{x},\bar{y})||. \tag{7.13}$$

n.	Caratteristiche	simb.	n.	Caratteristiche	simb.
1	Area (pixel)	$\|A\|$	14	Gradiente 1-2	G_{12}
2	Ascissa del Centro	$\bar{x}$	15	Gradiente 2-3	G_{23}
3	Ordinata del Centro	$\bar{y}$	16	Gradiente 3-4	G_{34}
4	Semiasse Maggiore	a	17	Valor Medio	T_{av}
5	Semiasse Minore	b	18	Valore Centrale	T_c
6	Angolo (gradi)	α	19	Rapporto 1	T_r
7	Diametro	dia	20	Rapporto 2	T_{cA}
8	Ellitticità	ell	21	2° Momento Totale	C_2
9	Rapporto tra le Aree	c_1	22	4° Momento Totale	C_4
10	Logaritmo dell'Area	c_2	23	Schiacciamento	E
11	Momento	M_1	24	Densità di Picco	D_p
12	Gradiente 1-4	G_{14}	25	Raggio Efficace	R_{ef}
13	Gradiente 1-3	G_{13}			

Fig. 7.12 - lista delle caratteristiche estratte.

Finora si è supposto di operare su una sola banda di luminosità. Nel caso di estrazione contemporanea su tre bande, gli algoritmi di *separazione* e *affinamento* vengono fatti operare su un'immagine mediata tra le tre frequenze che porterà ad un'unica lista di oggetti. Le prime 6 caratteristiche ($\bar{x}$, $\bar{y}$, a, b, α, $|A|$) che descrivono gli ellissi circoscritti agli oggetti saranno, di conseguenza, estratte una volta sola sull'immagine mediata. Le altre caratteristiche vengono invece estratte tre volte sulle tre frequenze. Ciò comporta che, l'estrazione su 3 bande, darà luogo non a 25=6+19 ma a 63=6+(19×3) parametri. La lista delle caratteristiche estratte è riassunta in figura 7.12 rispettando l'ordine di presentazione nel file generato dalla nostra procedura.

7.5 Esperimenti e confronti con SKICAT.

Prima di poter far funzionare i nostri algoritmi di valutazione su una qualsiasi lastra segmentata, è necessario fissare il valore di alcuni parametri. Per quanto riguarda la fase di *separazione*, occorre definire: il numero n di assi da ispezionare per la ricerca di doppi picchi (vedi equazione 7.6), il numero N di livelli entro cui limitare la luminosità (vedi equazione 7.7) e il numero k di settori da considerare per il calcolo delle medie (vedi algorimo di ricomposizione in figura 7.9). Per la fase di *affinamento* è necessario invece fissare le due soglie α e β richieste dall'algoritmo in figura 7.11.

n	*N*	*k*	*a*	*b*
5	40	4	0.1	0.5

Fig. 7.13 - parametri della fase di valutazione.

La tabella in figura 7.13 mostra i valori da attribuire ai suddetti parametri al fine di massimizzare il rapporto prestazioni/complessità (tali valori sono stati determinati a seguito di esperimenti su una grande varietà di oggetti). C'è da dire, comunque, che la sensibilità ai parametri è piuttosto bassa: variando i valori entro un *range* non troppo ampio, si provocano cambiamenti minimi nel funzionamento degli algoritmi.

Sfruttando le immagini di figura 7.14, analizzeremo ora il funzionamento degli algoritmi di valutazione. La figura 7.14a è un particolare del *footprint* già mostrato in figura 6.11: su questa immagine faremo *girare* gli algoritmi e mostreremo, volta per volta, i risultati intermedi. In figura 7.14b è evidenziata la maschera di segmentazione (già ottenuta nella fase precedente) dalla quale si può notare come la galassia centrale, essendo un oggetto molto brillante, tenda ad inglobare un gran numero di oggetti limitrofi. Sfruttando sia l'immagine che la maschera, l'algoritmo di *separazione* procede alla divisione (e alla successiva eventuale ricomposizione) degli oggetti multipli. Il risultato è mostrato in figura 7.14c dove tonalità di grigio diverse indicano oggetti diversi (è da notare che non c'è alcuna relazione tra le tonalità della figura 7.14b e quelle della 7.14c: nel primo caso infatti tonalità diverse corrispondono a diversi neuroni attivati). Guardando la figura 7.14c si noterà che, come avevamo predetto, gli oggetti ottenuti non sono affatto buone approssimazioni dei corpi celesti

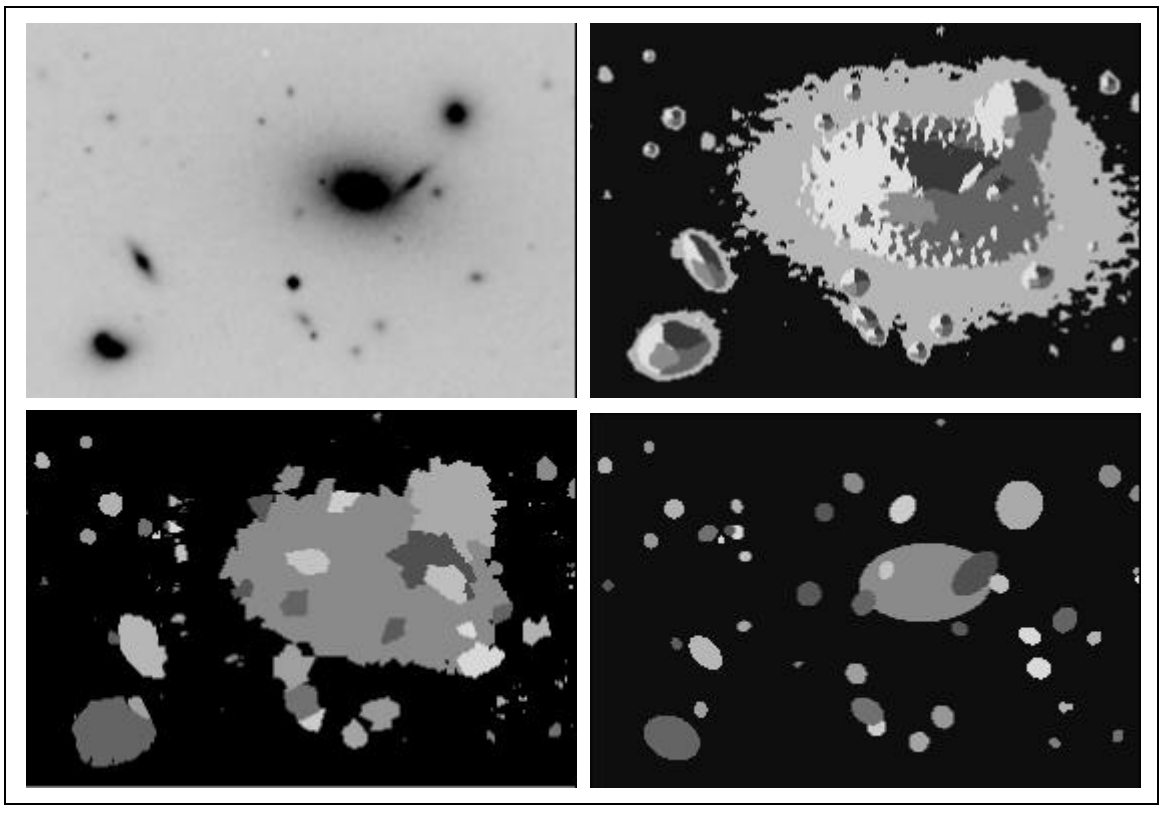

Fig. 7.14 - diverse fasi della valutazione degli oggetti dell'immagine in alto a sinistra.

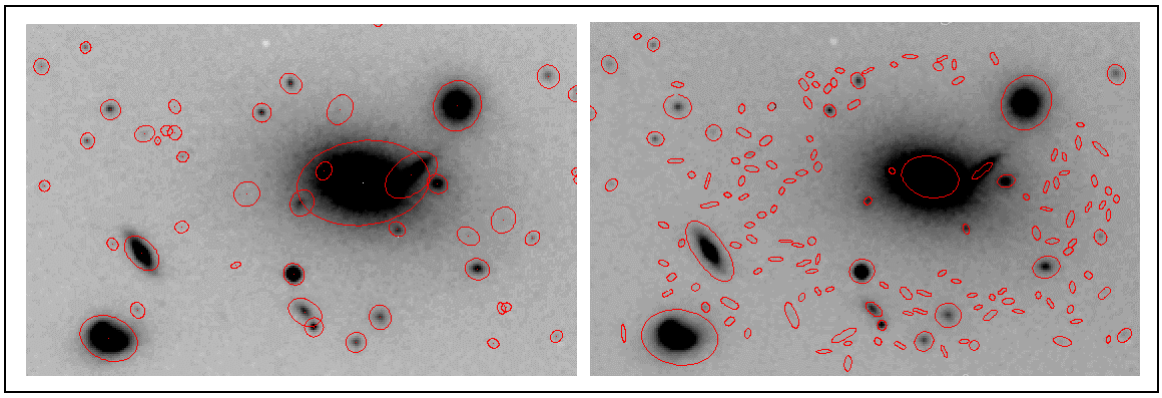

Fig. 7.15 - detezione nostra (a sinistra) e di SKICAT (a destra) sulla stessa immagine.

che si vogliono rappresentare. A questo scopo si introduce l'algoritmo di *affinamento* che avvicina il più possibile le forme degli oggetti a quelle dei corpi celesti (il risultato di questo *step* è in figura 7.14d). Sugli oggetti così ottenuti si estraggono le caratteristiche come discusso nel paragrafo precedente.

Se si sovrappongono all'immagine di figura 7.14a gli ellissi rappresentativi degli oggetti rilevati (si possono sfruttare, a tale proposito, le prime 6 caratteristiche estratte), si può fare una prima valutazione del risultato della detezione. Come si vede in figura 7.15a, a parte la presenza inevitabile di alcuni oggetti *spuri* (si veda a tale proposito la sezione dedicata agli sviluppi futuri), c'è una fedele rispondenza tra oggetti rilevati e corpi celesti presenti sulla lastra. Per evidenziare la bontà del nostro risultato, si mostra in figura 7.15b il risultato ottenuto da SKICAT sulla stessa immagine. Come si vede SKICAT, pur riconoscendo pressappoco gli stessi corpi, genera innumerevoli oggetti *spuri* (ricordiamo che un oggetto spurio è un oggetto a cui non corrisponde nessun corpo celeste) dovuti alla presenza dell'alone luminoso della galassia centrale che viene spezzettato in mille parti e non ricostruito. Un fatto del genere si ripercuoterà negativamente su tutti i programmi di analisi successivi.

Un confronto quantitativo tra i due metodi risulta piuttosto arduo. Una prima idea potrebbe essere quella di confrontare il numero di oggetti rilevati da ciascun metodo su una o piu lastre. Vincerebbe in questo caso il metodo che rileva più oggetti. Bisogna però tener conto che alcuni di questi oggetti potrebbero essere *spuri* e quindi essere privi di valore scientifico. Ciò ci porta ad un secondo sistema: limitare il confronto ai soli oggetti corretti (non *spuri*) rilevati. Un approccio del genere presuppone però l'esistenza di un termine di paragone ovvero di un catalogo corretto di una zona di cielo mediante il quale stabilire la correttezza di ogni oggetto rilevato. Cataloghi del genere esistono ma sono limitati a piccole zone di cielo, il nostro confronto quantitativo si limiterà quindi, solo per questa fase, ad una piccola immagine (se un'immagine 4096×2048 può essere considerata piccola). La zona di cielo in questione è il nucleo di un *cluster* di galassie conosciuto in letteratura come *Coma cluster* e il catalogo è quello generato da Lobo nel 1997 (vedi riferimenti) a partire da immagini CCD. La tabella in figura 7.16 mostra il risultato del confronto con SKICAT sul *Coma cluster* effettuato con una procedura automatica di *matching* tra i tre cataloghi.

	SKICAT	Il nostro metodo
Totale oggetti rilevati	2443	1923
Oggetti spuri	635	120
Oggetti corretti	1808	1803
Percentuale di correttezza	74.01	93.76

Fig. 7.16 - confronto con SKICAT sul nucleo del Coma cluster.

Come si può notare, la nostra percentuale di correttezza è nettamente superiore anche se SKICAT identifica qualche oggetto in più. Occorre però analizzare quest'ultimo dato alla luce del fatto che SKICAT genera tantissimi oggetti *spuri* e che il catalogo su CCD è molto più *profondo* (e quindi contiene molti più oggetti) rispetto agli altri due. Esiste quindi una probabilità non trascurabile che un oggetto *spurio* si venga a trovare nelle vicinanze di un oggetto del catalogo CCD non visibile (e quindi non rilevato) sulla nostra lastra fotografica. In questo caso avviene il *matching* e l'oggetto viene considerato, a torto, corretto. SKICAT genera più oggetti spuri di noi quindi ha una maggiore probabilità che ciò avvenga. Ciò si traduce in un lieve incremento ingiustificato del numero di oggetti rilevati.

In figura 7.17 mostriamo altri confronti qualitativi tra il nostro metodo e SKICAT su diverse lastre e con diverse tipologie di oggetti. Quando disponiamo di un catalogo di confronto, gli ellissi relativi ad oggetti certi vengono colorati in blu. In ogni caso il nostro metodo è vincente perchè, a parità di corpi celesti rilevati, aggiunge un numero di oggetti *spuri* minimo.

Riferimenti.

J. F. Jarvis, J. A. Tyson "FOCAS: Faint Object Classification And Analysis System" in *The Astronomical Journal*, vol. 86, no. 3, pp. 476-495, 1981.

C. Lobo, A. Bivano, F. Durret, D. Gerbal, O. Le Fevre, A. Mazure "A photometric catalogue of the Coma cluster core" in *Astronomy and Astrophysics*, no. 122, 1997.

A. S. Miller, M. J. Coe "Star/galaxy classification using Kohonen self-organizing maps" in *Mon. Mot. R. Astron. Soc.*, no. 279, pp. 293-300, 1996.

S. Odewahn, E. Stockwell, R. Pennington, R. Humphreys, W. Zumach "Automated Star/Galaxy Discrimination with Neural Networks" in *The Astronomical Journal*, vol. 103, no. 1, pp. 318-331, 1992.

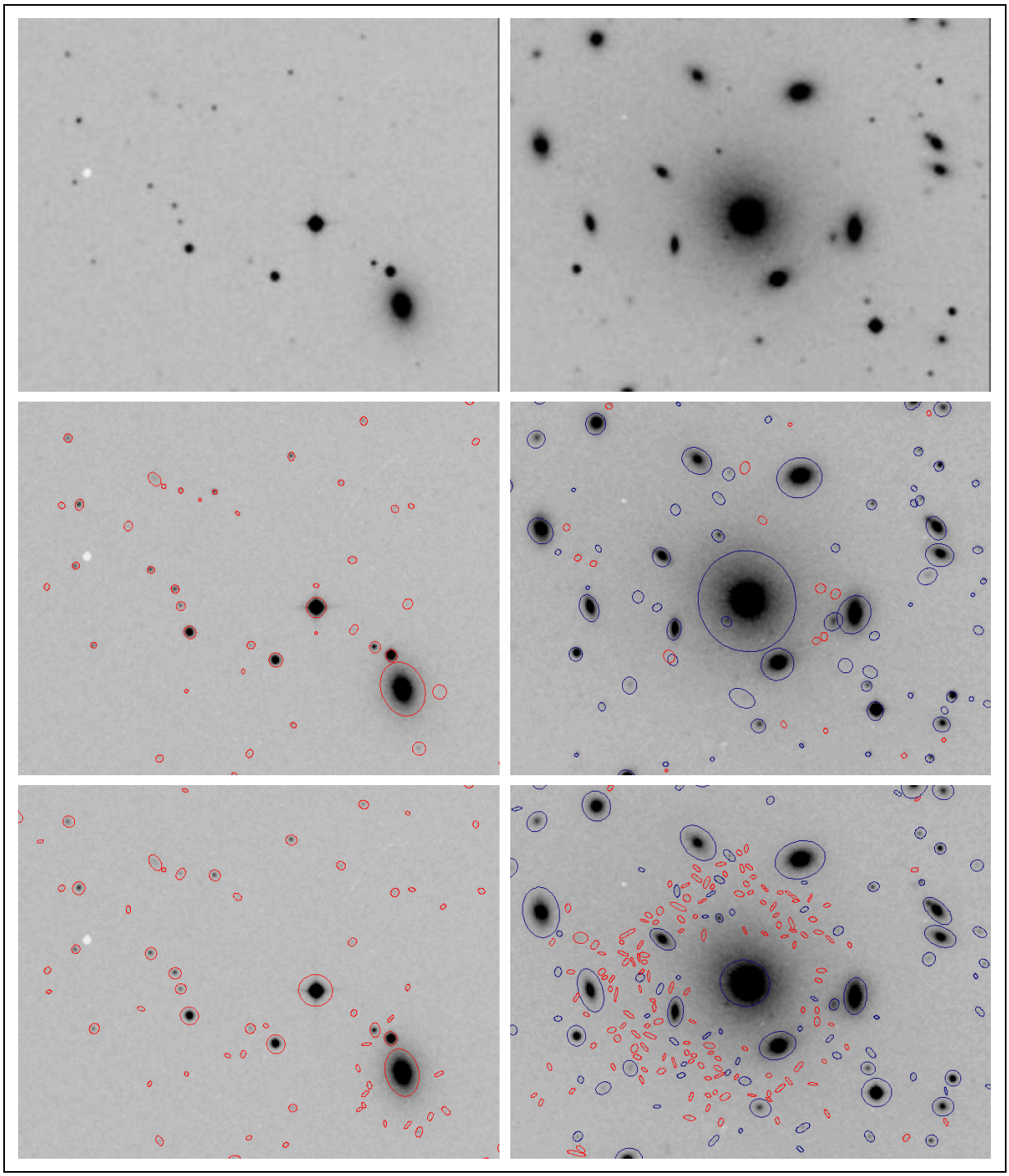

Fig. 7.17 - detezione nostra (al centro) e di SKICAT (in basso) sulle immagini in alto.

CAPITOLO 8

Terzo Passo: Classificazione

8.1 Introduzione.

Dopo aver identificato tutti gli oggetti presenti su una lastra e dopo aver valutato, per ciascuno di essi, una lunga serie di parametri fotometrici, ci si accinge ora a sviluppare dei criteri per la separazione tra stelle e galassie. Ogni oggetto verrà fatto cadere nella propria classe di appartenenza; solo dopo questa fase, sarà possibile effettuare futuri studi astrofisici. E' utile ricordare (lo si accennava già nel paragrafo 1.1) che il limite dell'utilizzabilità scientifica dei dati ricavati dalle *sky survey* è determinato proprio dall'accuratezza della separazione stelle/galassie. La percentuale di errore in questa fase non potrà superare il 10%, pena l'inutilizzabilità dei dati in molti programmi di analisi successivi (l'accuratezza “dichiarata” di SKICAT è appunto del 90%).

L'input della *classificazione* è costituito dal solo *file di caratteristiche* generato nella fase di *valutazione* degli oggetti. Si ricorda che questo file ha un record per ogni oggetto rilevato e un campo per ogni caratteristica estratta. Anche se le caratteristiche estratte sono in totale 25 (sintetizzate nella tabella di figura 7.7), utilizzeremo negli esperimenti solo le ultime 19 dato che le prime 6 ($|A|$, $\bar{x}$, $\bar{y}$, a, b, α) non hanno alcun valore discriminante e sono riportate solo per facilitare la visualizzazione grafica degli ellissi che circoscrivono gli oggetti. Lo stesso *file di caratteristiche*, dopo l'aggiunta di un nuovo campo che rappresenta la classe (S=stella, G=galassia) di ciascun oggetto, è restituito come output di questa fase.

Abbiamo sperimentato classificatori basati su reti neurali supervisionate e non supervisionate. Nel primo caso si fa apprendere la rete su un catalogo certo preesistente e poi la si applica al caso generale (si è gia parlato nel capitolo 2 delle capacità di generalizzazione delle reti neurali); nel secondo caso si lascia che la rete faccia emergere le classi valutando le similitudini esistenti tra gli oggetti (più precisamente tra i *pattern* che li rappresentano). Le reti supervisionate, come vedremo, garantiscono prestazioni migliori.

Prima di passare agli esperimenti è però necessario operare una selezione tra i parametri (caratteristiche) di cui disponiamo per snellire il lavoro del classificatore. Ci possono essere caratteristiche ridondanti o non discriminanti o caratteristiche che aggiungono addirittura del rumore. Eliminando le prime si può velocizzare il classificatore, eliminando le seconde si facilita l'apprendimento e si possono incrementare le prestazioni. La diminuzione del numero di caratteristiche si può quindi tradurre in un miglioramento della classificazione (si veda a questo proposito anche il problema *course of dimensionality* già esposto nel paragrafo 5.1).

Il paragrafo che segue è dedicato alla spiegazione di alcune tecniche di selezione delle caratteristiche che utilizzeremo per scegliere l'input del nostro classificatore. Mostreremo nel paragrafo 8.3 gli esperimenti relativi alla selezione dei parametri e nel paragrafo 8.4 i risultati di classificazione ottenuti sfruttando il sottoinsieme di parametri rivelatosi ottimale. I nostri risultati, come vedremo, sono migliori di quelli di SKICAT anche in questa fase.

8.2 Selezione delle caratteristiche.

Una delle tecniche più semplici per ridurre la dimensionalità dei pattern in input ad una rete neurale consiste nel selezionare un sottoinsieme di caratteristiche scartando le restanti (la PCA è una tecnica più complessa per risolvere lo stesso problema). Questo approccio (detto *selezione delle caratteristiche* o *feature selection*) è tipicamente usato quando tra gli input della rete vi sono caratteristiche con scarsa informazione o quando vi è una forte relazione tra sottoinsiemi di caratteristiche così che la stessa informazione è ripetuta più volte.

Un primo metodo di selezione, applicabile solo a reti supervisionate, consiste nello sfruttare la conoscenza appresa dalla rete stessa. Dopo aver fatto apprendere la rete su un training set composto dai pattern completi, si vanno ad analizzare le connessioni input-hidden: i neuroni input corrispondenti a caratteristiche poco discriminanti avranno i pesi delle connessioni prossimi allo zero poiché la rete ne ha dedotto la scarsa rilevanza e ha ritenuto opportuno, di conseguenza, indebolirne i segnali. La selezione avviene quindi scartando le caratteristiche con connessioni deboli.

Un secondo metodo di selezione, applicabile questa volta sia a reti supervisionate che non supervisionate, consiste nel valutare le prestazioni di classificazione per tutti i possibili sottoinsiemi di caratteristiche. Il sottoinsieme che, a parità di training set, ci assicura i risultati migliori è il sottoinsieme discriminante ottimale. L'unico intoppo è che questo metodo è *time consuming* dato che per n caratteristiche siamo costretti a far apprendere la rete 2^n volte su 2^n diversi insiemi di caratteristiche. Dato che la scelta delle caratteristiche avviene una sola volta, il gioco potrebbe valere la candela se non esistessero tecniche *greedy* che, a parità di efficacia, hanno tempi computazionali notevolmente inferiori.

Una di queste tecniche è la *sequential search tecnique* che determina ogni nuovo sottoinsieme di caratteristiche in base al comportamento della rete sui sottoinsiemi

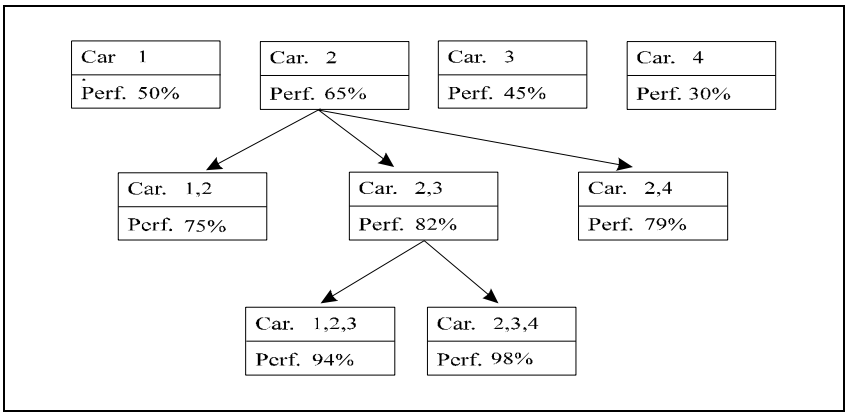

Fig. 8.1 - esempio di sequential forward selection.

precedenti. Esistono due diversi approcci a questa tecnica detti rispettivamente *sequential forward selection* e *sequential backward elimination*.

La ***sequential forward selection*** considera, in prima istanza, una caratteristica per volta selezionando quella che ha dato migliori risultati di classificazione. Ad ogni passo successivo viene addizionata una caratteristica e rivalutato il criterio di selezione come mostra la figura 8.1. Questa tecnica non tiene conto, purtroppo, che l'informazione congiunta di due o più caratteristiche può essere superire alla somma delle informazioni contenute nelle caratteristiche se prese singolarmente. La figura 8.2 mostra un esempio di distribuzione di pattern, appartenenti a due diverse classi, su uno spazio di caratteristiche bidimensionale. Ciascuna delle due caratteristiche, se considerata singolarmente, non è discriminante (dà luogo ad un grosso *overlapping* tra le classi) ma, se considerata in congiunzione con l'altra, dà luogo ad una netta separazione.

Per ovviare a questo inconveniente, la ***sequential backward elimination*** parte considerando l'intero insieme di caratteristiche ed elimina una caratteristica per volta (vedi figura 8.3). Si applica il criterio di selezione su tutti i sottoinsiemi che si ottengono eliminando dal primo insieme una sola caratteristica e si considera solo il sottoinsieme che risulta più discriminante. Il processo si ripete su questo sottoinsieme

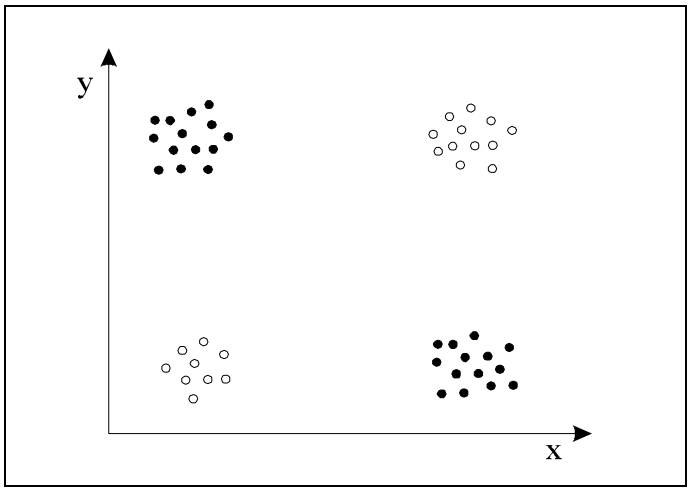

Fig. 8.2 - informazione congiunta di due caratteristiche.

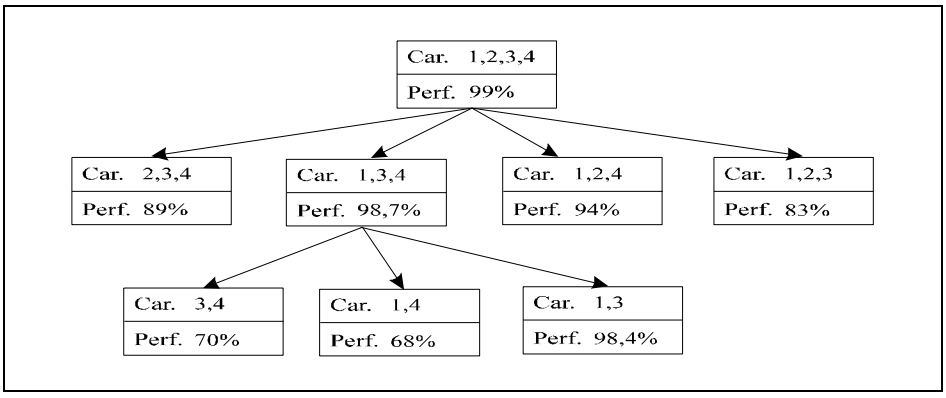

Fig. 8.3 - esempio di sequential backward elimination.

fino a quando non si ottiene un numero di caratteristiche adeguato che garantisce una buona classificazione. Pur risolvendo il problema dell'informazione congiunta, questo metodo impiega un maggior tempo di computazione dato che vengono fatti più training su pattern ad alta dimensionalità.

8.3 Esperimenti di selezione delle caratteristiche.

Per la ricerca del sottoinsieme ottimale di caratteristiche abbiamo pensato di costruire un "piccolo" training set di 1000 oggetti significativi (scelti tra stelle e galassie di dimensioni e tipi diversi) di cui si conosce l'esatta classificazione. Dopo aver estratto per questi oggetti le 25 caratteristiche come descritto nel paragrafo 7.4, si può procedere ad una *sequential backward elimination* utilizzando reti supervisionate e non supervisionate su un test set composto da altri 1000 oggetti. Abbiamo preferito un piccolo training set per non appesantire troppo l'apprendimento delle reti (che deve essere ripetuto molte volte su sottoinsiemi diversi di caratteristiche) dato che, in questo frangente, non ci interessa ottenere una classificazione molto raffinata ma solo adeguata ad evidenziare le caratteristiche discriminanti.

La rete neurale non supervisionata che utilizziamo nel test è una *Neural-Gas* non gerarchica con 20 neuroni output. Assumiamo che il *labeling* delle classi sia manuale: alla fine di ogni training, si calcola, per ogni neurone *i* della rete, il numero di stelle s_i e il numero di galassie g_i del training set che cadono nella regione di Voronoi da *i* rappresentata. Per s_i maggiore di g_i si etichetta *i* come stella; per s_i minore di g_i si etichetta *i* come galassia. In realtà non si può parlare più di apprendimento non supervisionato dato che si presuppone di conoscere le etichette dei pattern del training set. Dimostreremo che, anche con questa facilitazione (quindi a maggior ragione per il caso generale), le reti non supervisionate non danno buoni risultati.

La rete neurale supervisionata scelta per l'esperimento è una
MLP con un livello hidden composto da 20 neuroni e con algoritmo di apprendimento *scaled coniugate gradient* (vedi paragrafo 3.5). Si concede alla rete un solo neurone di output che assumerà un valore compreso nell'intervallo [0,1]. Il responso viene arrotondato all'intero più vicino e si assume 0 per "stella" e 1 per "galassia".

È necessario, a questo punto, spendere qualche parola in più sull'apprendimento delle reti in questo e nei successivi esperimenti. Avremmo potuto utilizzare i pattern così come ci provenivano dal *file di caratteristiche* ma, purtroppo, l'intervallo dei valori possibili varia molto da caratteristica a caratteristica. Il parametro *ell* varia, per esempio, tra 0 e 1; la densità di picco D_p supera di frequente quota 30000. Si impone quindi una normalizzazione degli input per dare ad ogni caratteristica lo stesso peso. Scegliamo di normalizzare i pattern calcolando, per ogni caratteristica *i*, media E_i e varianza V_i sul training set (nell'assunzione che il training set sia rappresentativo dell'universo) e creando, per ogni pattern x che si dà in pasto alla rete, il corrispondente normalizzato x' in questo modo:

$$x' = (x - E)/V \text{ dove } E = (E_1,\ldots,E_{25}) \text{ e } V = (V_1,\ldots,V_{25}).$$

La tabella in figura 8.4 mostra i passi successivi di *backward elimination* per la rete non supervisionata e le corrispondenti percentuali di correttezza. Il migliore risultato si raggiunge, come mostra anche il grafico in figura 8.5, per il sottoinsieme che comprende le sole caratteristiche *dia*, M_1, G_{13}, T_r e T_{cA}. Il risultato migliore supera di poco l'85% di correttezza.

Caratteristiche eliminate	**Distorsione**	**Errori**	**Corr. (%)**
Nessuna	2.51991	178	83.21
Densità di Picco (D_p)	2.52115	179	83.12
Ellitticità (*ell*)	2.36895	194	81.74
Gradiente 2-3 (G_{23})	2.31752	185	82.57
Logaritmo dell'area (c_2)	2.24	183	82.57
Valore centrale (T_c)	2.20618	190	82.11
Rapporto tra le Aree (c_1)	1.95012	200	81.19
4° Momento Totale (C_4)	1.74341	185	82.57
2° Momento Totale (C_2)	1.64549	185	82.57
Gradiente 3-4 (G_{34})	1.55388	176	83.40
Gradiente 1-2 (G_{12})	1.50692	173	83.67
Valor Medio (T_{av})	1.44025	172	83.77
Raggio Efficace (R_{ef})	1.3251	169	84.04
Schiacciamento (*E*)	0.9382	169	84.04
Gradiente 1-4 (G_{14})	0.87717	157	85.15
Rapporto 2 (T_{cA})	0.76993	159	84.96
Rapporto 1 (T_r)	0.59958	175	83.49
Diametro (*dia*)	0.46143	173	83.67
Momento (M_1)	0.10891	291	72.81
Gradiente 1-3 (G_{13})	-	-	-

Fig. 8.4 - risultati della backward elimination per la rete non supervisionata.

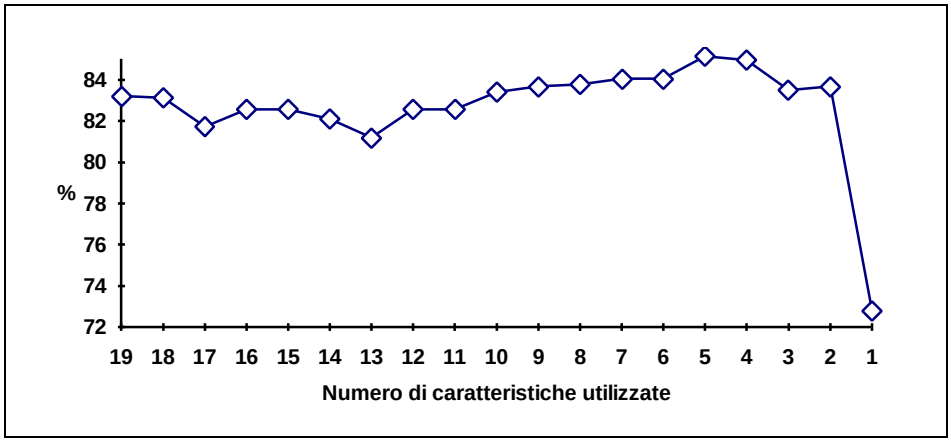

Fig. 8.5 - andamento della correttezza per le reti non supervisionate.

Per capire le cause del fallimento delle reti non supervisionate sono state studiate le distribuzioni dei pattern di training scegliendo due componenti per volta. Per quasi tutte le coppie di caratteristiche considerate sono emerse nette separazioni tra stelle e galassie solo a basse magnitudini (per oggetti quindi molto brillanti). Man mano che si sale con la magnitudine, la separazione diveniva sempre più impercettibile fino a scomparire in un un fitto nucleo di oggetti diversi (in figura 8.6 è mostrato, come esempio, la distribuzione sulle caratteristiche T_c e G_{13}).

Le difficoltà di apprendimento della rete non supervisionata non devono quindi meravigliare dato che, dovendo individuare dei cluster nello spazio di caratteristiche, la rete non può far altro che inglobare erroneamente tutto il nucleo in una sola classe impoverendo, di conseguenza, le prestazioni.

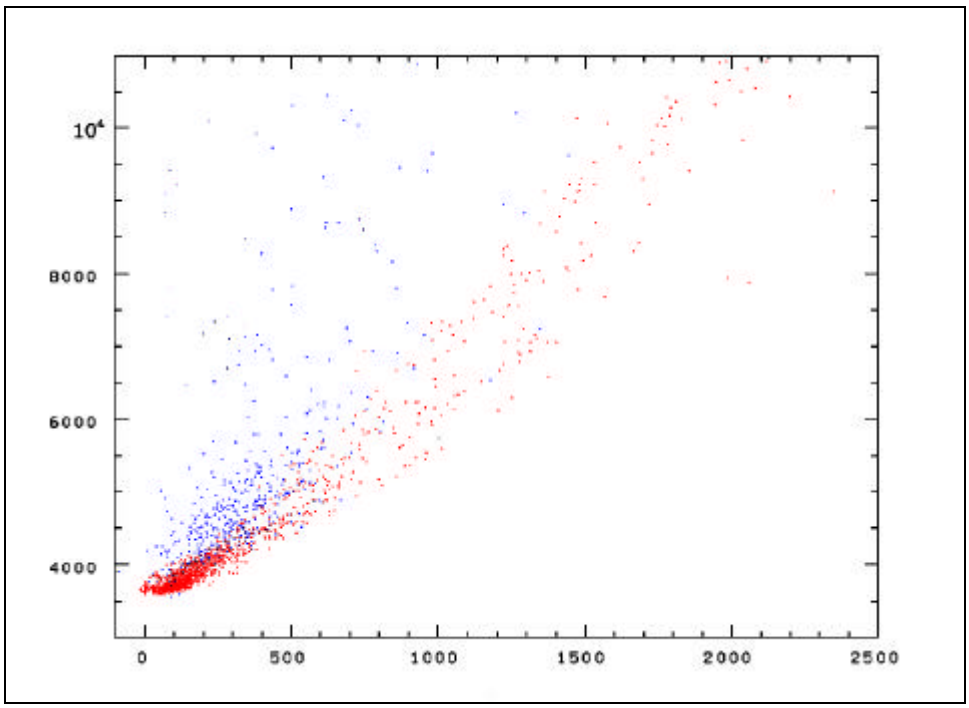

Fig. 8.6 - distribuzione sulle caratteristiche G_{13} (asse *x*) e T_c (asse *y*).

Ben diversi sono i risultati ottenuti dalla rete MLP che riesce ad individuare dei semispazi che discriminano in modo molto più accurato gli oggetti, anche ad alte magnitudini. Applicando anche a questa rete la *backward elimination*, si ottengono i passi di eliminazione mostrati nella tabella in figura 8.7. La prima cosa che salta all'occhio è che la percentuale di correttezza, in alcuni casi, supera anche il 95%. L'andamento di questa percentuale per passi successivi (come mostra il grafico in figura 8.8) è piuttosto fluttuante a causa del numero limitato di cicli (1000) che abbiamo concesso alla rete. Il *trend* si mantiene comunque pressoché costante per le prime 10 iterazioni per poi iniziare una fase di decrescita che prosegue fino all'ultima iterazione: in questo caso una diminuzione del numero di caratteristiche non si risolve mai (a parte minime fluttuazioni) in un miglioramento delle prestazioni. Analizzando questi dati, decidiamo di utilizzare, per il training delle reti, le seguenti 10 caratteristiche: c_2, G_{13}, G_{12}, G_{23}, T_r, T_{cA}, C_2, E, D_p, R_{ef} che corrispondono al punto di inversione del *trend* (da costante a decrescente). In questo modo conserviamo la maggiore informazione possibile minimizzando il tempo di training (in questo esperimento abbiamo utilizzato solo 1000 pattern e 1000 cicli di apprendimento ma, in genere, non sarà così).

Caratteristiche eliminate	**Funz. err.**	**Errori**	**Corr. (%)**
Nessuna	0.411	48	95,2
4° Momento Totale (C_4)	0.42	47	95,3
Rapporto tra le Aree (c_1)	0.462	51	94,9
Diametro (*dia*)	0.391	48	95,2
Momento (M_1)	0.429	49	95,1
Valor Medio (T_{av})	0.427	51	94,9
Ellitticità (*ell*)	0.410	47	95,3
Gradiente 3-4 (G_{34})	0.481	53	94,7
Gradiente 1-4 (G_{14})	0.441	52	94,8
Valore centrale (T_c)	0.435	48	95,2
Rapporto 2 (T_{cA})	0.444	55	94,5
2° Momento Totale (C_2)	0.465	58	94,2
Raggio Efficace (R_{ef})	0.453	58	94,2
Logaritmo dell'area (c_2)	0.491	65	93,5
Gradiente 1-3 (G_{13})	0.501	63	93,7
Schiacciamento (E)	0.503	75	92,5
Gradiente 2-3 (G_{23})	0.52	81	91,9
Rapporto 1 (T_r)	0.553	76	92,4
Densità di Picco (D_p)	0.565	99	90,1
Gradiente 1-2 (G_{12})	-	-	-

Fig. 8.7 - risultati della backward elimination per la rete supervisionata.

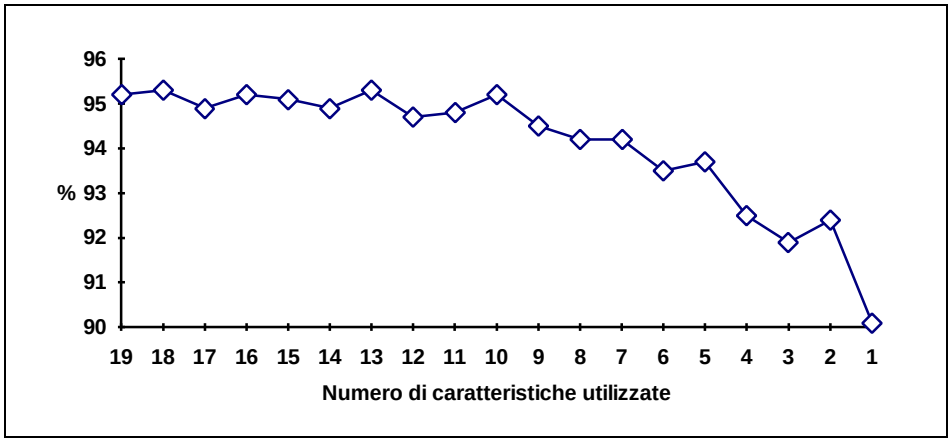

Fig. 8.8 - andamento della correttezza per le reti supervisionate.

8.4 Esperimenti di classificazione e confronti con SKICAT.

Una volta stabilito che le reti non supervisionate non sono adatte alla classificazione di corpi celesti (per quanto detto nel paragrafo precedente), non ci rimane che effettuare una serie di esperimenti per vedere quale, tra le reti supervisionate del capitolo 3, offre le prestazioni migliori e quale, quindi, utilizzeremo nel nostro programma.

La tabella in figura 8.9 mostra, a tale proposito, i risultati di alcuni test effettuati sul training set “piccolo” del paragrafo precedente (epurato delle caratteristiche non discriminanti) in termini di errore medio raggiunto dopo 1000 iterazioni e di percentuali di correttezza calcolate su un test set di altri 1000 oggetti.
Le reti migliori sono risultate quelle con funzioni di apprendimento *scaled coniugate gradient* e *quasi Newton method* anche se, nel secondo caso, il tempo di computazione è stato molto elevato a causa della lentezza dell'algoritmo *line search* (vedi paragrafo 3.4). La nostra scelta è quindi caduta sull'algoritmo *scaled coniugate gradient*.

Per quanto concerne la dimensione della rete, una serie di esperimenti effettuati sul solito training set “piccolo”, ci ha portati a preferire uno strato hidden con 20 neuroni. La soluzione migliore per classificare oggetti provenienti da campi stellari è quindi, riassumendo quanto detto finora, la seguente:

Algoritmo di apprendimento	Funz. err.	Tempo (sec.)	Corr. (%)
Gradiente discendente batch	0.541204	856	93
Gradiente discendente serial	0.524621	1135	94.2
Coniugate Gradient	0.423015	3924	94.4
Scaled Coniugate Gradient	0.41947	1804	95.7
Quasi Newton Method	0.400832	4174	95.9

Fig. 8.9 - risultati della backward elimination per la rete supervisionata.

- estrarre da ogni oggetto le seguenti 10 caratteristiche: c_2, G_{13}, G_{12}, G_{23}, T_r, T_{cA}, C_2, E, D_p, R_{ef} (vedi tabella 7.7);
- normalizzare le caratteristiche estratte sottraendo la media e dividendo per la varianza (media e varianza sono calcolate sul training set);
- classificare i pattern così ottenuti con una MLP con 20 neuroni hidden, 1 neurone output e algoritmo di apprendimento *scaled coniugate gradient* fatta apprendere in precedenza su un training set rappresentativo dell'universo.

Ciò stabilito, possiamo passare a considerare un training set più impegnativo ricavato da un catalogo di una parte del *Coma cluster* (un cluster di galassie di cui abbiamo già parlato nel paragrafo 7.5) generato nel 1981 da Godwin (vedi riferimenti). Questo catalogo (centrato in $RA = 12^h57^m18^s$; $Dec = 28°14'24"$) si estende per più di 2° per lato e corrisponde ad una lastra di circa 8000×8000 pixel. Abbiamo frammentato questa lastra in 16 *footprint* 2048×2048 e abbiamo dato ciascuno di questi in pasto al nostro programma limitandoci alle prime due fasi (segmentazione ed estrazione delle caratteristiche).

Abbiamo ottenuto come output 16 *file di caratteristiche* che abbiamo confrontato con il catalogo in questione eliminando tutti gli oggetti non presenti nel secondo e determinando, per i rimanenti, la classe di appartenenza. Un gran numero di oggetti deboli è stato eliminato in questa fase dato che Godwin non va molto in profondità (il catalogo è stato ottenuto con metodi di analisi/ispezione manuale). Nonostante ciò abbiamo scelto Godwin perché lo si può considerare un catalogo certo e, in questa fase, abbiamo bisogno di classificazioni certe (si noti che al passo precedente sceglivamo per i confronti Lobo che, pur essendo carente come correttezza di classificazione, va molto più a fondo nella detezione).

Gli oggetti rimasti a questo punto (circa 8000 tra stelle e galassie) sono stati ripartiti in un training set di 5000 elementi ed un test set di 3000 elementi. Sul training set così generato è stata fatta apprendere in 4000 cicli la nostra MLP che ha raggiunto un errore medio pari a 0.38564 in poco più di 2 ore di lavoro. La rete è stata sfruttata per classificare i 3000 elementi del test set ottenendo i risultati mostrati nella tabella di figura 8.11 dove si mostrano anche le prestazioni di SKICAT per gli stessi oggetti. Le nostre prestazioni sono migliori di esattamente il 6.5%.

Come abbiamo visto nel paragrafo 7.5, SKICAT ha una correttezza di detezione del 74.01%. Se componiamo questo risultato con quello di classificazione, abbiamo che solo il 66.42% degli oggetti rilevati da SKICAT ha valore scientifico (oggetti corretti e correttamente classificati) contro il 90.24% di quelli rilevati dal nostro approccio. È un bel passo avanti.

	SKICAT	**Il nostro metodo**
Totale oggetti	3000	3000
Classificazioni corrette	2692	2887
Classificazioni errate	308	113
Percentuale di correttezza	89.75	96.25

Fig. 8.11 - confronto con SKICAT sul Coma cluster.

Riferimenti.

C. M. Bishop “Neural Networks for Pattren Recognition” *Oxford University Press*, 1995.

J. G. Godwin, N. Metcalfe, J. V. Peach “A catalogue of magnitudes, colours, ellipticities and position angles for 6724 galaxies in the field of the Coma cluster” in *Mon. Not. R. Astron. Soc.* no. 202, 1981.

CONCLUSIONI E SVILUPPI FUTURI

In questo lavoro di Tesi ci si proponeva di costruire un software in grado di ottenere prestazioni migliori di SKICAT sia nell'identificazione che nella classificazione di oggetti celesti. Entrambi gli obbiettivi sono stati centrati e i risultati sono più che soddisfacenti infatti:

- il 94% degli oggetti identificati sono corretti se confrontati con cataloghi ottenuti da CCD mentre SKICAT non supera il 74% di correttezza: il 26% degli oggetti generati da SKICAT è quindi *spurio* contro il 6% dell'approccio proposto;
- la percentuale di correttezza della classificazione stelle/galassie sfiora quota 96.3% contro SKICAT che si attasta invece intorno al 90%.

Componendo i due risultati si può notare che solo il 66% degli oggetti rilevati da SKICAT ha valore scientifico (si tratta di oggetti corretti e correttamente classificati) contro il 90% di quelli rilevati dall'approccio proposto.

Numerose soluzioni originali sono state proposte in questo lavoro di tesi tra cui:

- l'organizzazione dei neuroni su più strati sovrapposti (paragrafo 4.6) e l'applicazione di un algoritmo di clustering sull'output (paragrafo 4.7) al fine di automatizzare la fase di *labeling* di una generica rete neurale non supervisionata;
- l'applicazione della funzione tangente iperbolica sull'output della PCA in fase di segmentazione (paragrafo 6.3) per identificare anche gli oggetti più deboli presenti su una lastra;
- la creazione di un algoritmo intelligente di separazione/ricomposizione (paragrafo 7.2) che evita l'*oversplitting* di oggetti multipli e di un algoritmo di affinamento (paragrafo 7.3) che rende più affidabile l'estrazione delle caratteristiche.

Le prestazioni ottenute dall'approccio proposto, pur essendo molto buone, sono suscettibili di ulteriori miglioramenti sia in fase di detezione che in fase di classificazione:

- per migliorare la percentuale di correttezza della detezione è auspicabile la creazione di un algoritmo (del resto già in fase avanzata di progettazione) in grado

di riconoscere ed eliminare automaticamente i soli oggetti *spuri* presenti su una lastra;

- per velocizzare l'apprendimento delle reti supervisionate per la classificazione si può pensare a un training *ibrido* che sfrutti le potenzialità di più algoritmi in contemporanea.

Per quanto riguarda il secondo punto, è stato notato che, utilizzando l'algoritmo di apprendimento *scaled coniugate gradient*, l'errore medio diminuisce velocemente fino a un certo livello, superato il quale, tende a stabilizzarsi. Il *metodo di Newton* ha iterazioni molto più lente ma, di contro, riesce a raggiungere valori più bassi dell'errore medio. Si può pensare quindi di ibridare i due algoritmi utilizzando il primo fin quando l'errore medio scende per poi sfruttare il secondo a partire dalla configurazione di pesi finale del primo. In questo modo si riuscirebbe a scendere più in basso sulla funzione di errore senza una perdita eccessiva di tempo.

BIBLIOGRAFIA

S. Andreon, S. Zaggia, R. de Carvalho, S. Djorgovski *et al.* "The CRONA (Caltech-Roma-Napoli) project" in *astro-ph/*9706238, accettato in giugno 1997.

R. Beale, T. Jackson "Neural Computing: An Introduction" *Institute of Physics Publishing Bristol and Philadelphia*, 1980.

C. M. Bishop "Neural Networks for Pattren Recognition" *Oxford University Press*, 1995.

A. De Luca, M. Ricciardi "Introduzione alla Cibernetica" *Franco Angeli Editore*, 1986.

B. Everitt "Cluster Analysis" *Social Science Research Council. Heinemann Educational Books. London*, 1977.

U. M. Fayyad, S.G. Djorgovski, N. Weir "From Digitized Images to On-line Catalogs: A Machine Learning Application" in *AI Magazine*, accettato in aprile 1996.

B. Fritzke "Growing Cell Structures - A Self-Organizing Network for Unsupervised and Supervised Learning" in *Neural Networks*, vol. 7, no. 9, pp. 1441-1460, 1994.

J. G. Godwin, N. Metcalfe, J. V. Peach "A catalogue of magnitudes, colours, ellipticities and position angles for 6724 galaxies in the field of the Coma cluster" in *Mon. Not. R. Astron. Soc.* no. 202, 1981.

J. F. Jarvis, J. A. Tyson "FOCAS: Faint Object Classification And Analysis System" in *The Astronomical Journal*, vol. 86, no. 3, pp. 476-495, 1981.

J. Karhunen, J. Joutsensalo "Generalizations of Principal Component Analysis, Optimization Problems and Neural Networks" in *Neural Networks* vol. 8, no. 4, pp. 549-562, 1995.

J. Koh, M. Suk, S. Bhandarkar "A Multilayer Self-Organizing Feature Map for Range Image Segmentation" in *Neural Networks*, vol. 8, no. 1, pp. 67-86, 1995.

T. Kohonen "Self-organized formation of topologically correct feature maps" in *Biological Cybernetics* vol. 43, pp. 59-69, 1982.

C. Lobo, A. Bivano, F. Durret, D. Gerbal, O. Le Fevre, A. Mazure "A photometric catalogue of the Coma cluster core" in *Astronomy and Astrophysics*, no. 122, 1997.

L. Macera "Reti Neurali e Pattern Recognition - finalmente un cervello elettronico?" in *MC-Microcomputer*, no. 102, pp. 210-213, 1990.

T. Martinetz, S. Berkovich, K. Schulten "Neural-Gas Network for Vector Quantization and its Application to Time-Series Prediction" in *IEEE transactions on Neural Networks*, vol. 4, no. 4, pp. 558-568, 1993.

A. Mazzetti "Intelligenza e Vita" *Apogeo-Editrice di Informatica*, 1994.

A. S. Miller, M. J. Coe "Star/galaxy classification using Kohonen self-organizing maps" in *Mon. Mot. R. Astron. Soc.*, no. 279, pp. 293-300, 1996.

S. Odewahn, E. Stockwell, R. Pennington, R. Humphreys, W. Zumach "Automated Star/Galaxy Discrimination with Neural Networks" in *The Astronomical Journal*, vol. 103, no. 1, pp. 318-331, 1992.

E. Oja, J. Karhunen, L. Wang, R. Vigario "Principal and Independent Components in Neural Networks - Recent Developments" al *7th Italian Workshop on Neural Networks*,1995.

E. Oja, H. Ogawa, J. Wangviwattana "Learning in Nonlinear Contrained Hebbian Networks" in *Artificial Neural Networks* pp. 385-390, 1991.

N. Pal, S. Pal "A Review on Image Segmentation Techniques" in *Pattern Recognition*, vol. 26, no. 9, pp. 1277-1294, 1993.

D. Rumelhart, G Hinton, R.Williams "Learning Internal Representations by Error Propagation" in *ICS Report 8506 Institute of Cognitive Science University of California*, 1985.

N. Weir, U. M. Fayyad, S. G. Djorgovski "Automated star/galaxy classification for digitized POSS-II" in *The Astronomical Journal*, vol. 109, no. 6, pp. 2401-2414, 1995.

N. Weir, U. M. Fayyad, S. G. Djorgovski, J. Roden "The SKICAT System for Processing and Analyzing Digital Imaging Sky Surveys" in *Publications of the Astronomical Society of the Pacific*, no. 107, pp. 1243-1254, 1995.