Università degli Studi di Trieste

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI
Corso di Laurea in Fisica

Tesi di laurea

Studio di una memoria associativa con pretrattamento dei dati

Candidato:
Giulio Alessandrini
Matricola 62200376

Relatore:
Professor Marco Budinich

Anno Accademico 2008-2009

Sommario

In questo lavoro mi propongo di analizzare il comportamento e le capacità di un modello di rete neurale - le cosiddette memorie associative o reti di Hopfield - nella memorizzazione di dati in formato binario. [3]

Per rendere il lavoro di analisi più comprensibile ho scelto di utilizzare le ventisei lettere dell'alfabeto latino. Le immagini sono in piccolo formato per agevolare la fase di simulazione.

Abstract

In this work I will anaylize the behavior and capacity of a Hopfiel Net – a model of neural network also known as Associative Memory. The net will be trained in order to memorize binary data. [3]

For the sake of clarity and immediacy of comprehension I chose as examples the twentysix latin alphabet's letters in a small, black-and-white rapresentation made by myself.

Indice

Notazione **V**

1 Neuroni umani e neuroni simulati **1**
1.1 Un'introduzione biologica 1
1.2 Primi passi con la simulazione 2

2 Memorie associative **4**
2.1 Costruzione e memorizzazione di un esempio 4
2.2 Generalizzazione 5
2.3 Simulazione 8
2.4 Il problema della correlazione 10
2.5 Prova di memoria 10
2.5.1 Analisi preliminare 10
2.5.2 Dati rumorosi o parzialmente errati 13

3 Trattamento dei dati **16**
3.1 Analisi delle componenti principali 16
3.2 Decomposizione ai valori singolari 17

4 Torniamo alle memorie **19**

A L'energia nelle memorie associative **24**

Ringraziamenti **26**

Bibliografia **27**

Notazione

In questo scritto farò largo uso di matrici e vettori. Per rendere più semplice la lettura riporto qui un breve schema delle notazioni utilizzate

simbolo	significato	rango
M	una matrice	2
M_i	la colonna i-esima di una matrice	1
M_{ij}	un elemento di matrice	0
$\mathbf{v}$	un vettore	1
$\mathbf{v}_\nu$	un vettore tra molti	1
v_i	una componente di un vettore	0
$v_{\nu i}$	una componente di uno tra più vettori	0
x	un numero	0

Per fortuna non ci sono insiemi di matrici...

Capitolo 1

Neuroni umani e neuroni simulati

1.1 Un'introduzione biologica

Il cervello umano contiene in media 10^{11} neuroni ed ognuno di essi si collega con un migliaio di suoi vicini, questo significa che in un volume di circa 1400 cm^3 sono presenti 10^{14} connessioni. È grazie a questo impressionante apparato che il nostro cervello riesce a gestire la complessità degli stimoli che ci giungono continuamente dal mondo esterno, così come le funzioni vitali e quella particolarità degli organismi superiori — o forse solo dell'uomo — che chiamiamo autocoscienza.

La chiave di volta di questo meccanismo sono le cellule nervose: i *neuroni*. La struttura del neurone può essere schematizzata abbastanza semplicemente (ai fini di questo lavoro) in:

- un blocco centrale, il *corpo cellulare*;
- dei filamenti che si estendo dal centro, i *dendriti*;
- un'appendice più o meno lunga detta *assone*;
- le *sinapsi nervose*, delle protuberanze poste al termine dell'assone che si collegano ai dendriti degli altri neuroni.

Ogni neurone è sede di un potenziale elettrico regolato da una serie di ioni (sodio, potassio e cloro) che può subire variazioni, in entrambi i sensi, grazie a dei neurotrasmettitori rilasciati dalle sinapsi. Se questo potenziale supera una certa soglia, la membrana cellulare diventa permeabile al sodio, il potenziale si innalza lungo tutto l'assone e causa il rilascio dei neurotrasmettitori che agiscono sui neuroni vicini. Terminata questa fase (detta *firing* in inglese) si verifica un periodo refrattario e quindi la cellula è pronta per una nuova trasmissione.

Pur con questi pochi elementi, a metà del secolo scorso alcuni ricercatori pensarono che un'approccio basato sulle simulazioni al calcolatore potesse gettare luce sulle dinamiche principali del cervello. Si cominciò quindi ad elaborare e mettere alla prova dei modelli matematici di neuroni (e di reti di neuroni) che potessero mimare il funzionamento cerebrale e permetterne uno studio più accurato. Molto presto, tuttavia, ci si rese conto che queste *reti neurali* rappresentavano un nuovo strumento di calcolo e risoluzione dei problemi ancora tutto da investigare, indipendentemente dal successo della ricerca in campo biologico. Il motivo è che la struttura altamente parallela del nostro cervello lo rende differente da un elaboratore elettronico sotto molti punti di vista:

- riesce a funzionare anche in seguito al danneggiamento o alla disconnessione di alcune sue parti;
- è in grado di apprendere e adattarsi agli stimoli esterni (recente la scoperta che i neuroni a specchio, che ricreano una copia dell'ambiente esterno, possono funzionare basandosi solo sui dati uditivi nelle persone cieche dalla nascita);
- è in grado di gestire informazione approssimativa o incompleta.

I nostri elaboratori più moderni sono ancora lontani da queste prestazioni ma hanno comunque un'importante vantaggio: il tempo tipico di un neurone è di 10^{-3} secondi contro i 10^{-6} di un microprocessore al silicio. Queste sono le ragioni che spingono per realizzare una macchina con le prestazioni di una rete neurale biologica e la velocità dei processori più moderni.

1.2 Primi passi con la simulazione

Nel 1943, McCulloch e Pitts pongono le basi delle reti neurali artificiali con l'introduzione del loro modello di neurone [4]. Si tratta di un neurone binario che esegue la somma di tutti gli n segnali in ingresso $x_{i=1,\dots,n}$ moltiplicati per il peso w_i della connessione da cui provengono e, a qualora il valore così ottenuto superi una certa soglia ϑ, produce un segnale in uscita. Questo comportamento si può ottenere con la formula

$$y = \Theta\left(\sum_{i=1}^{n} w_i x_i + \vartheta\right) \tag{1.1}$$

in cui Θ, detta *funzione di trasferimento*, vale uno nel semiasse non negativo e zero altrimenti (si tratta della funzione a gradino).

Si può ottenere una notazione più compatta immaginando di mettere tutti i segnali in ingresso in un vettore a n componenti (compreso un ulteriore

ingresso x_0 che vale sempre 1 e il cui peso sia ϑ per tenere conto della soglia) e utilizzando l'operazione di prodotto scalare:

$$y = \Theta\left(\mathbf{w} \cdot \mathbf{x}\right) \tag{1.2}$$

Questa notazione getta nuova luce sul funzionamento di questo tipo di neurone: si può immaginare che lo spazio a n dimensioni sia tagliato dall'iperpiano di equazione $\mathbf{w} \cdot \mathbf{x} = 0$ che funge da spartiacque, questo sistema classifica i vettori in base alla loro posizione rispetto a quest'iperpiano.

L'ultima parte da chiarire resta la scelta del vettore dei pesi $\mathbf{w}$. Qui ci viene in aiuto una regola proposta nel 1949 da Donald Hebb: il collegamento tra due neuroni si rafforza quando essi sono attivi contemporaneamente, altrimenti si indebolisce. Nella nostra notazione l'intensità del collegamento tra i neuroni i e j si può scrivere:

$$w_{(ij)} \leftarrow \ w_{(ij)} + \varepsilon y_i y_j \tag{1.3}$$

a condizione di sostituire il valore in uscita 0 con -1 per tenere conto dell'indebolimento delle connessioni; questo si può ottenere con una funzione di trasferimento che vale -1 sull'asse negativo (si può ottenere con $2\vartheta(x) - 1$, che è la funzione segno).

Capitolo 2

Memorie associative

2.1 Costruzione e memorizzazione di un esempio

Il soggetto di questo studio, le reti di Hopfield, possono essere immaginate come un insieme di neuroni come quelli definiti nel capitolo precedente in cui ogniuno è connesso a tutti gli altri [2]. Come vedremo questo tipo di collegamento presenta caratteristiche che ricordano la memoria biologica, motivo per cui queste reti sono anche conosciute come *memorie associative*. In questo caso la distinzione tra valori in ingresso e in uscita è più sfumata e non granché importante, scriveremo quindi

$$s_i' = \operatorname{sgn}(W_i \cdot \mathbf{s}) \tag{2.1}$$

in cui W è ora una matrice $n \times n$ che contiene i pesi per tutte le possibili connessioni ed $\mathbf{s}$ è un vettore che rappresenta lo stato di tutti i neuroni (i quali, ricordiamo, sono binari e possono assumere i valori ± 1). Se immaginiamo che la funzione segno agisca su tutte le componenti, l'aggiornamento complessivo si scrive

$$\mathbf{s}' = \operatorname{sgn}(W \cdot \mathbf{s}) \tag{2.2}$$

A questo punto indichiamo con $\boldsymbol{\xi}$ lo stato che vogliamo far memorizzare alla nostra rete e utilizzando la regola di apprendimento hebbiano poniamo i pesi proporzionali alla correlazione tra i neuroni

$$W_{ij} = \gamma \xi_i \xi_j$$

con la condizione $w_{ii} = 0$.

Bisogna ora verificare che lo stato $\boldsymbol{\xi}$ sia stato effettivamente memorizzato cioè sia uno stato stazionario. Per farlo, mettiamolo nella rete e vediamo

cosa succede:

$$\begin{aligned}\xi_i' &= \operatorname{sgn}(W_i \cdot \boldsymbol{\xi}) \\ &= \operatorname{sgn}\left(\sum_{i=1}^{n} W_{ij}\xi_j\right) \\ &= \operatorname{sgn}\left(\sum_{i=1}^{n} (\gamma\xi_i\xi_j)\xi_j\right) \\ &= \operatorname{sgn}(\gamma n \xi_i)\end{aligned}$$

a questo punto basta porre $\gamma = 1/n$ per ottenere

$$\xi_i' = \operatorname{sgn}(\xi_i)$$

che è esattamente quello che cercavamo.

Per uno stato generico $\mathbf{s}$ posso supporre che esistano l componenti uguali a, e quindi $n-l$ componenti diverse da, lo stato $\boldsymbol{\xi}$. Senza perdita di generalità posso ordinarle in modo da avere

$$\begin{aligned}s_i' &= \operatorname{sgn}(w_i \cdot \mathbf{s}) \\ &= \operatorname{sgn}\left(\frac{1}{n}\sum_{i=1}^{n} w_{ij}s_j\right) \\ &= \operatorname{sgn}\left(\frac{1}{n}\sum_{i=1}^{l}(\xi_i\xi_j)\xi_j + \frac{1}{n}\sum_{i=l}^{n}(\xi_i\xi_j)(-\xi_j)\right) \\ &= \operatorname{sgn}(\frac{l-(n-l)}{n}\xi_i) \\ s_i' &= \operatorname{sgn}([2l-n]\xi_i)\end{aligned}$$

In questo caso il risultato è diverso a seconda che $\mathbf{s}$ sia più o meno vicino allo stato $\boldsymbol{\xi}$. Infatti, se i due vettori si trovano dalla medesima parte dell'iperpiano identificato da $\mathbf{w}$ (cioè se $0 < \mathbf{s} \cdot \boldsymbol{\xi} < 1$) lo stato convergerà verso $\boldsymbol{\xi}$; nella condizione opposta, verso $-\boldsymbol{\xi}$. Lo spazio delle configurazioni è quindi diviso in due bacini di attrazione della medesima misura.[1]

2.2 Generalizzazione

Adesso che siamo riusciti a memorizzare un esempio viene spontaneo chiedersi se sia possibile memorizzarne altri, altrimenti queste reti sarebbero certamente

[1] Qui c'è da fare una precisazione per quanto riguarda la frontiera: se la funzione segno è descritta in modo che $\operatorname{sgn}(0) = 1$, essa appartiene al bacino di attrazione di $\boldsymbol{\xi}$, per $\operatorname{sgn}(0) = -1$ appartiene a quello di $-\boldsymbol{\xi}$.

poco interessanti. Una strada scontata è applicare nuovamente la regola di Hebb e definire i pesi con m esempi

$$W_{ij} = \frac{1}{n}\sum_{\nu=1}^{m} \xi_{\nu i}\xi_{\nu j}$$

Di nuovo è possibile utilizzare una notazione semplificata mettendo tutti gli $\boldsymbol{\xi}_\nu$ in una matrice Ξ di dimensione $n \times m$. Allora la matrice dei pesi si può scrivere come

$$W = \frac{1}{n}\Xi \cdot \Xi^T \tag{2.3}$$

Una volta preparata la rete bisogna verificare, come nel caso precedente, la stabilità degli esempi memorizzati; partiamo da un generico esempio $\boldsymbol{\xi}_\mu$

$$\begin{aligned}
\xi'_{\mu i} &= \operatorname{sgn}\left(W_i \cdot \boldsymbol{\xi}_\mu\right) \\
&= \operatorname{sgn}\left(\sum_{j=1}^{n} W_{ij}\xi_{\mu j}\right) \\
&= \operatorname{sgn}\left(\frac{1}{n}\sum_{j=1}^{n}(\sum_{\nu=1}^{m}\xi_{\nu i}\xi_{\nu j})\xi_{\mu j}\right) \\
&= \operatorname{sgn}\left(\frac{1}{n}\sum_{j=1}^{n}\left[\xi_{\mu i}\xi_{\mu j}\xi_{\mu j} + (\sum_{\nu\neq\mu}\xi_{\nu i}\xi_{\nu j})\xi_{\mu j}\right]\right) \\
&= \operatorname{sgn}\left(\xi_{\mu i}\left[1 + \frac{\xi_{\mu i}}{n}\sum_{j=1}^{n}\sum_{\nu\neq\mu}\xi_{\nu i}\xi_{\nu j}\xi_{\mu j}\right]\right)
\end{aligned}$$

L'ultimo passaggio è reso possibile dal fatto che $\xi_{\mu i}\xi_{\mu i} = 1$. Affinché questo punto sia stabile occorre che

$$\xi_{\mu i}\sum_{j=1}^{n}\sum_{\nu\neq\mu}\xi_{\nu i}\xi_{\nu j}\xi_{\mu j} > -n \tag{2.4}$$

Il problema non è più risolubile esattamente, bensì in maniera probabilistica: immaginiamo che tutti gli addendi della (2.4) siano variabili aleatorie con probabilità p di valere 1 e $1-p$ di valere -1. La sommatoria è formata da $n(m-1)$ termini; per cui se k di questi valgono 1 il risultato sarà $k(1) + [n(m-1) - k](-1) = 2k - n(m-1)$. Per semplificare la trattazione, a questo punto conviene porre $p = \frac{1}{2}$ perché il fattore $\xi_{\mu i}$ davanti alla sommatoria scambia p con $1-p$ a seconda del segno.[2] Ricapitolando la

[2] Quest'ipotesi significa chiedere che la correlazione tra, e all'interno degli, esempi sia nulla. Ne vedremo le conseguenze nel prossimo capitolo.

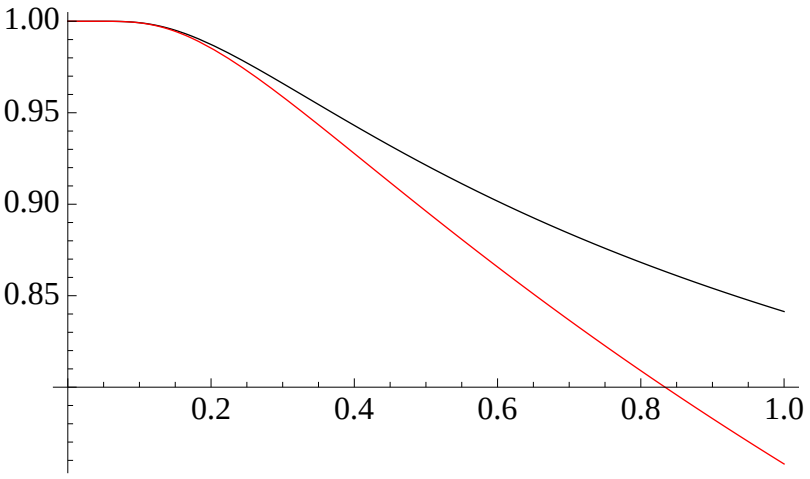

Figura 2.1: La stabilità di un punto in funzione del rapporto $\frac{m}{n}$. In nero l'integrale (2.5) e in rosso l'approssimazione in serie al primo ordine.

condizione (2.4) significa che la probabilità che un punto sia stabile è

$$\sum_{l=-n+1}^{n(m-1)} P\left[\sum_{j=1}^{n}\sum_{\nu\neq\mu}\xi_{\nu i}\xi_{\nu j}\xi_{\mu j} = l\right]$$

con

$$P\left[\sum_{j=1}^{n}\sum_{\nu\neq\mu}\xi_{\nu i}\xi_{\nu j}\xi_{\mu j} = l\right] = P(l) = \binom{n(m-1)}{l}\frac{1}{2}^{n(m-1)}$$

Questa è una distribuzione binomiale e per quello che ci interessa complica un po' le cose. Se facessimo tendere n ed m all'infinito però, la distribuzione si potrebbe approssimare con la corrispondente gaussiana di parametri $\mu = 0$ e $\sigma = \sqrt{n(m-1)} \simeq \sqrt{mn}$. [3] Con questo accorgimento le equazioni precedenti diventano

$$\int_{-n}^{\infty} dx G_{\mu\sigma}(x) \simeq= \frac{1}{2}\left(\text{erf}\left(\sqrt{\frac{1}{2\alpha}}\right) + 1\right) \tag{2.5}$$

$$G_{\mu\sigma}(x) \propto e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

in cui abbiamo posto $\alpha = \frac{m}{n}$. Per dare un senso a questo risultato conviene espandere in serie la funzione erf

$$\mathrm{P}(\alpha) = 1 - e^{-\frac{1}{2\alpha}}\sqrt{\frac{\alpha}{2\pi}}$$

[3] Ricordiamo che già per un numero di termini relativamente piccolo ($nm \sim 10$) l'approssimazione gaussiana non porta a grandi errori.

Figura 2.2: Alcune delle immagini utilizzate nella simulazione.

Se vogliamo che tutti gli n punti del nostro esempio siano stabili al 99% – una condizione ragionevole – bisogna imporre

$$\left(1 - e^{-\frac{1}{2\alpha}}\sqrt{\frac{\alpha}{2\pi}}\right)^n > \frac{99}{100}$$
$$e^{-\frac{1}{2\alpha}} > \frac{1}{n100}$$
$$\alpha < \frac{1}{2\log(100n)}$$

Per portare a termine il conto in modo analitico abbiamo fatto delle approssimazioni considerando solo il termine esponenziale e supponendo α piccolo.

Infine, se ci interessa che siano stabili tutti i punti di tutti gli esempi dobbiamo imporre

$$\alpha < \frac{1}{4\log(100n)} \tag{2.6}$$

Tutto questo conto è servito a dare un idea delle condizioni per la stabilità statica degli esempi memorizzati. Siccome questa rete assume un comportamento dipendente dal tempo è, però, possibile che una piccola instabilità iniziale conduca a una serie crescente di errori. Uno studio della dinamica del sistema porta a concludere che per $\alpha > 0.138$ svanisce ogni effetto di memoria [1].

2.3 Simulazione

Armati dei risultati della sezione precedente possiamo cominciare a simulare la nostra rete. Per semplicità gli esempi binari che ho scelto sono cento immagini in bianco e nero di lettere dell'alfabeto latino (alcune sono riportate nella figura 2.2). Per ridurre i tempi di simulazione le ho disegnate con 20×20 punti. Questo significa che $n = 400$ e, per quanto visto prima (2.6), serve $m \leq 9$ per l'equilibrio statico (che in questo caso verifica anche $\alpha < 0.138$)

Come prima cosa conviene mettere alla prova la rete inserendo come stato gli esempi per verificare che siano effettivamente presenti nella memoria; in altre parole si segue quanto esposto nella sezione 2.2 dal punto di vista della simulazione. Prima di mettersi a scrivere a capofitto il codice è però utile fare alcune piccole considerazioni sulla regola (2.1). Infatti non è possibile calcolare simultaneamente l'aggiornamento di tutti i neuroni poiché ognuno risente del collegamento con tutti gli altri. L'unico modo per simulare una dinamica di questo tipo è il metodo stocastico: scelto un neurone a caso,

lo si aggiorna e quindi si passa al successivo finché la rete non si stabilizza. L'implementazione diretta di questo procedimento è, pero, molto inefficiente perché ogni volta che viene selezionato un neurone che non necessita di aggiornamento la rete compie un giro a vuoto. Si può risparmiare tempo generando una lista dei neuroni da aggiornare e scegliendo (sempre in modo casuale) fra di essi. Un primo modo potrebbe essere trovare le posizioni dei ± 2 nel vettore

$$\operatorname{sgn}(W \cdot \mathbf{s}) - \mathbf{s},$$

un secondo, quello da me utilizzato, andare a caccia dei -1 in

$$\mathbf{s} \operatorname{sgn}(W \cdot \mathbf{s});$$

in cui il prodotto è da intendersi componente per componente.

Un altro accorgimento utile riguarda il prodotto scalare $W \cdot \mathbf{s}$, la cui esecuzione richiede $n(n-1)$ somme ed n^2 moltiplicazioni. Supponiamo che il neurone j-esimo abbia cambiato segno in seguito all'aggiornamento, il nuovo stato S' sarà allora

$$\mathbf{s}' = \mathbf{s} - (0, \ldots, 2S_j, \ldots, 0);$$

se poi, applicando nuovamente la regola (2.1), cambia stato il neurone k-esimo

$$\begin{aligned} \mathbf{s}''_k &= \operatorname{sgn}(W_k \cdot \mathbf{s}') \\ &= \operatorname{sgn}(W_k \cdot [\mathbf{s} - (0, \ldots, 2S_j, \ldots, 0)]) \\ &= \operatorname{sgn}(W_k \cdot [\mathbf{s} + (0, \ldots, 2S'_j, \ldots, 0)]) \\ &= \operatorname{sgn}(W_k \cdot \mathbf{s} + W_k \cdot (0, \ldots, 2S'_j, \ldots, 0)). \end{aligned}$$

Questo significa che basta fare una volta sola il dispendioso prodotto $W \cdot \mathbf{s}$ e poi aggiornarlo con l'equazione appena trovata.

Per disporre di un adeguato controllo sulla rete ho deciso per un approccio graduale, composto dai seguenti passi:

1. selezionare un insieme di m esempi a caso;
2. generare con questi la matrice dei pesi con l'equazione (2.3);
3. preso a caso uno degli m esempi, inserirlo nella rete
   - preparare la lista con i neuroni che devono cambiare stato
   - aggiornarne uno
   - proseguire finché restano neuroni da aggiornare
4. proseguire come al punto 3 mettendo alla prova tutti gli esempi.

Ho ripetuto questo algoritmo diverse volte con un solo esempio, quindi, verificata la bontà del codice, sono passato a reti con $m > 1$ esempi. Questo tipo di approccio può essere applicato a piacimento e offre il vantaggio di fornire molta statistica al prezzo di lunghi tempi di simulazione.

2.4 Il problema della correlazione

Per avere un'idea del risultato di questo primo conto, ho realizzato degli istogrammi che mostrano la distribuzione degli errori (figura 2.3). Si nota subito come comincino a comparirne alcuni già con $m = 3$, indice di qualche problema molto serio. Possiamo dare una spiegazione qualitativa di questo comportamento: nel trattare l'equazione (2.4) avevamo supposto la probabilità per ogni punto di essere bianco o nero come $\frac{1}{2}$; in realtà il colore di un punto in un immagine è molto correlato con quello dei punti vicini, inoltre la correlazione si estende a punti corrispondenti di immagini diverse. Proponendo due esempi molto *vicini* la rete memorizza uno stato intermedio, cosa facilmente verificabile con le immagini. Ecco cosa ottiene quando si cercano di memorizzare le lettere A, B e C:

Per non incappare in questo inconveniente dovremmo utilizzare immagini di questo tipo

che ovviamente non hanno alcun significato per un essere umano.

Se non vogliamo rinunciare a queste reti bisogna trattare questi esempi in modo da eliminare la correlazione, una possibile strada sarà esplorata nel capitolo 3.

2.5 Prova di memoria

2.5.1 Analisi preliminare

Pur gravemente limitati dal problema della correlazione, possiamo cercare di capire più a fondo la dinamica delle memorie associative.

Cominciamo con il notare che l'evoluzione di una rete con i pesi definiti dall'equazione (2.3) corrisponde alla discesa lungo il gradiente della *funzione di Lyapunov* del sistema (c'è un conto dettagliato nell'appendice A). Ovviamente noi speriamo che i minimi assoluti della funzione siano in corrispondenza degli stati che vogliamo memorizzare — e se rimaniamo sotto i limiti trovati nella sezione precedente dovremmo averne la certezza — ma nulla si può dire su eventuali minimi relativi, che, infatti, sono presenti in gran numero e contribuiscono alla creazione di stati attrattori spuri.

Il modo in cui avviene la discesa verso il minimo può dare già alcune informazioni preliminari. Come si evince dalla figura 2.4, il numero di passi

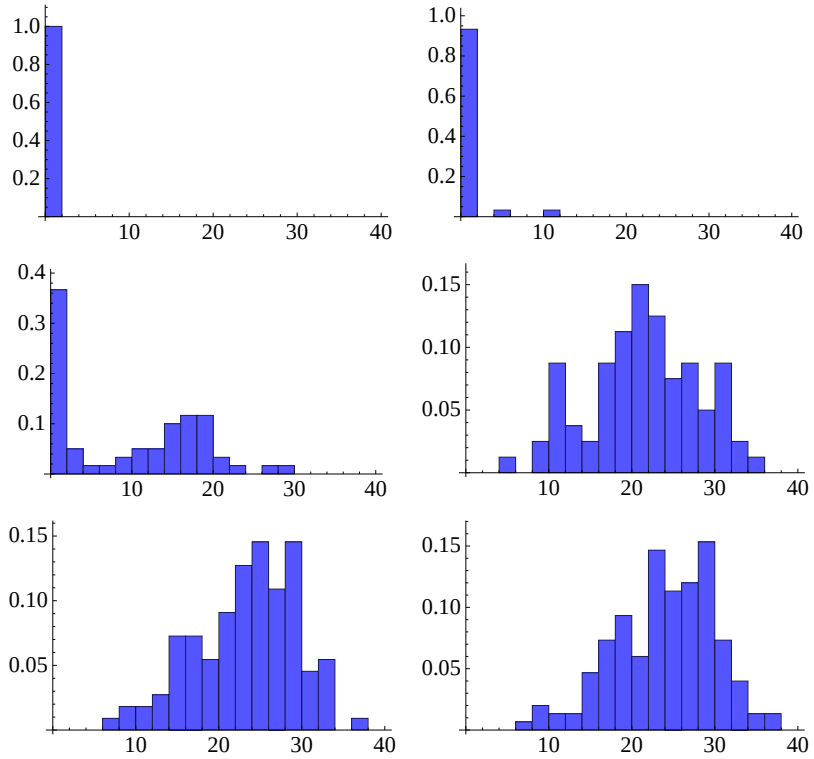

Figura 2.3: I risultati della prima simulazione con 1, 3, 6, 8, 11 e 15 lettere nella memoria. In orizzontale la percentuale di errore (ogni canale corrisponde all'1%); in verticale la probabilità misurata.

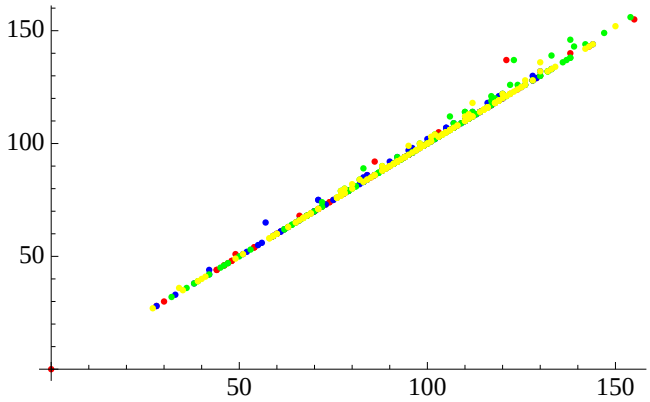

Figura 2.4: Passi necessari per portare lo stato in ingresso al più vicino punto di stabilità in funzione della distanza da quest'ultimo. Rosso, blu, verde e giallo corrispondono a $m = 9, 11, 13, 15$.

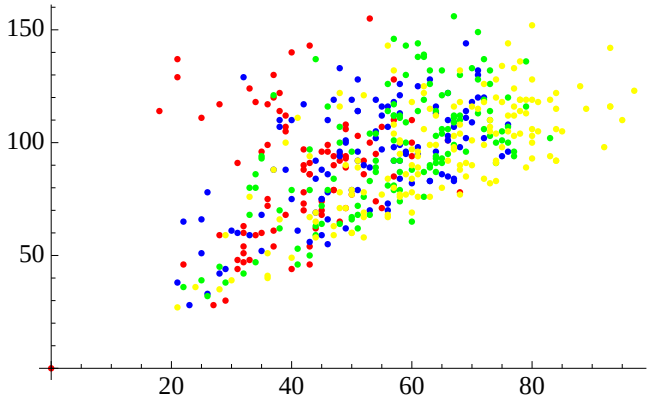

Figura 2.5: Grafico analogo al 2.4 ma con in ascissa il numero dei neuroni fuori posto alla prima iterazione.

compiuti dalla rete corrisponde quasi sempre alla distanza (in termini di bit sbagliati) dello stato di partenza dall'esempio memorizzato. Questo fatto non è per nulla scontato, vista la complessità dello spazio.

Se ripensiamo a come viene eseguita la simulazione possiamo sfruttare il fatto che i neuroni da aggiornare non sono scelti a caso, bensì da una lista che è aggiornata ad ogni iterazione; questo potrebbe fornire un atro indicatore da tenere d'occhio. La figura 2.5 mostra i passi compiuti in funzione della lunghezza della lista alla prima iterazione; rispetto alla 2.4, anche se in questo caso la correlazione lineare è molto meno marcata, si vede comunque una dipendenza dal numero di esempi.

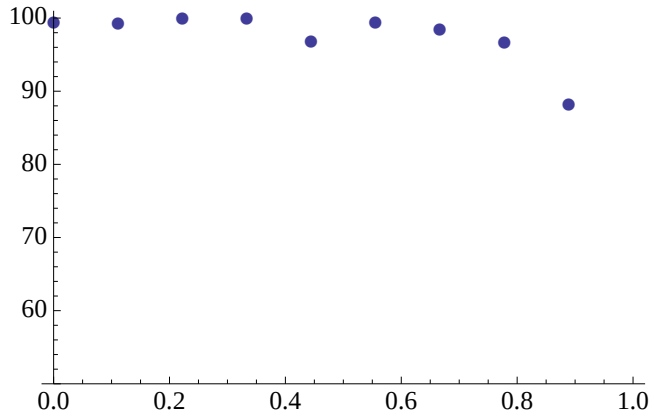

Figura 2.6: Percentuale di recupero di un'immagine rispetto al coefficiente di rumore.

2.5.2 Dati rumorosi o parzialmente errati

Come già detto nell'introduzione, le memorie biologiche sono in grado di ricordare partendo da informazioni incomplete o errate (fino ad un certo limite); è possibile verificare che anche le reti di Hopfield possiedono questa caratteristica. Cominciamo con il chiedere il riconoscimento di esempi rumorosi, come questo

Dopo diverse prove ho potuto constatare che le prestazioni della rete non decrescono linearmente, bensì manifestano un piccolo calo oltre la soglia di rumore del 60% per poi crollare oltre l'80% (figura 2.6). Si tratta comunque di prestazioni di tutto rispetto, considerando che pochi tra noi saprebbero dire al primo colpo che le immagini seguenti

sono una A, una I e una U; e in questo caso il coefficiente di rumore è solo (!) 0.5 .

Un altro problema potrebbe essere la perdita di un intero segmento di informazione alla inizio o alla fine di una comunicazione. Proviamo a vedere come la rete risponde a uno stato di questo tipo

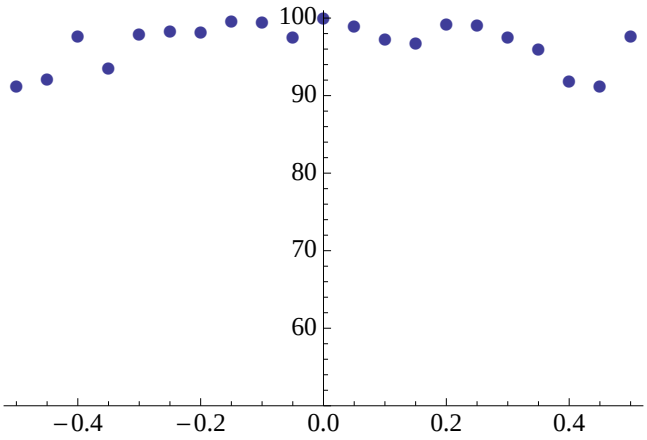

Figura 2.7: Percentuale di recupero di un'immagine rispetto ad una frazione mancante (i numeri negativi significano una rimozione dalla fine del vettore di 400 elementi che rappresenta l'immagine).

Nella figura 2.7 si può vedere come la percentuale di recupero decresca in modo approssimativamente quadratico con il crescere dei pezzi mancanti all'inizio (numeri positivi) e alla fine (numeri negativi) del vettore di 400 elementi che rappresenta l'immagine.

Un'ultima prova interessante prende spunto dal mondo biologico: dato un oggetto, gli esseri umani lo riconoscono facilmente anche dopo che questo è stato traslato o ruotato. Siccome queste memorie funzionano per somiglianza, ci si può aspettare un certo grado di tolleranza a fenomeni di questo tipo. Per rendere le operazioni più semplici ho fatto una prova con le traslazioni

In questo caso i risultati sono simili al caso precedente ma gli errori crescono più rapidamente (figura 2.8).

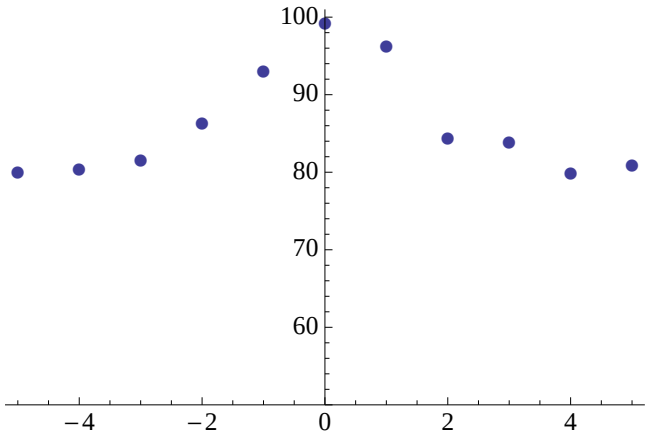

Figura 2.8: Percentuale di recupero di un'immagine rispetto ad uno spostamento a destra (semiasse positivo) e a sinistra (semiasse negativo).

Capitolo 3

Trattamento dei dati

3.1 Analisi delle componenti principali

Lo scopo di questo breve capitolo è trovare un metodo efficiente per ridurre la correlazione dei dati che dovranno essere gestiti dalla nostra memoria. L'idea di partenza è che in un insieme di dati con molte componenti la maggior parte dell'informazione può essere concentrata in poche direzioni fondamentali; si cercano allora delle nuove variabili maggiormente significative attraverso la combinazione lineare delle variabili di partenza. Per queste ragioni questo procedimento prende il nome di *analsi delle componenti principali.*[1]

Dato un insieme di valori $\mathbf{x} = (x_1, \ldots, x_n)$ vogliamo quindi delle nuove variabili del tipo

$$y_i = \mathbf{k}_i \cdot \mathbf{x},$$

che catturino gli aspetti principali dei dati, dobbiamo quindi trovare una rotazione degli assi che massimizzi la varianza

$$\langle y_i^2 \rangle - \langle y_i \rangle^2 = \langle \sum_j k_j x_j \rangle^2 - \left(\sum_j k_j \langle x_j \rangle \right)^2 .$$

Si possono rendere i conti più semplici ponendo

$$\langle \mathbf{x} \rangle = 0 \text{ per ogni } j$$

che si può ottenere sottraendo ad ogni componente del vettore la sua media, e restare alle prese con

$$\langle \sum_j k_j x_j \rangle^2 = \langle \sum_j \sum_l k_j x_j k_l x_l \rangle = \sum_{j,l} k_j C_{jl} k_l,$$

[1] Nel caso in esame di immagini binarie da 20×20 punti lo spazio delle configurazioni ha 400 dimensioni. Tenendo presente la grande correlazione che sussiste tra le immagini, è facile pensare che questo spazio non sia densamente popolato dagli esempi (oltretutto c'è posto per 2^{400} stati!)

in cui C è la matrice di correlazione dei dati e, nel nostro caso particolare di dati a media nulla, anche la matrice delle covarianze.

Per trovare il massimo di questa funzione si può usare il metodo dei moltiplicatori di Lagrange, imponendo l'ulteriore condizione $\mathbf{k} \cdot \mathbf{k} = 1$, che ci assicura che il massimo della varianza di y dipenda solo dalla direzione di k è non dal suo modulo. L'equazione da massimizzare diventa

$$\mathbf{k} \cdot C \cdot \mathbf{k} - \lambda(\mathbf{k} \cdot \mathbf{k} - 1).$$

Derivando rispetto a $\mathbf{k}$ e ponendo uguale a zero si ottiene

$$C \cdot \mathbf{k} - \lambda \mathbf{k} = 0$$

che, escludendo la soluzione banale $\mathbf{k} \equiv 0$, ci porta a concludere che λ è un autovalore di C e il vettore $\mathbf{k}$ un suo autovettore.

Si puo dimostrare facilmente che le altre $n - 1$ componenti principali sono i restanti autovalori di C. Allora, le nuove variabili che stiamo cercando possono essere scritte come

$$y_i = E_i \cdot \mathbf{x}$$

o più in generale

$$\mathbf{y} = E^T \cdot \mathbf{x},$$

dove abbiamo chiamato E la matrice degli autovettori di C.

Dalle equazioni precedenti possiamo anche dedurre che gli autovalori di C rappresentano la varianza dei dati nell'autospazio corrispondente, infatti

$$\langle y_i \rangle = E_i^T \cdot C \cdot E_i = \lambda_j.$$

3.2 Decomposizione ai valori singolari

L'utilizzo del sistema appena descritto passa per la ricerca degli autovettori di una matrice; questo metodo è il più elegante da dimostrare ma piuttosto esoso in fatto di risorse: bisogna trovare le radici di un polinomio di grado n e risolvere un sistema di n equazioni. Anche se la simmetria della matrice di correlazione ci assicura che gli autovalori siano reali, in un conto numerico di tale portata gli errori si accumulano e rendono molto difficile raggiungere risultati ottimali.

Un'alternativa interessante è rappresentata dalla *decomposizione ai valori singolari* (o SVD in inglese), un metodo più moderno e molto più stabile dal punto di vista computazionale. Lo scopo di questo metodo è scomporre una matrice M di dimensioni $m \times n$ nel prodotto

$$M = U \cdot L \cdot V^* \tag{3.1}$$

in cui:

- U è una matrice unitaria $m \times m$ le cui colonne sono autovettori di $M \cdot M^*$;
- L è una matrice diagonale $m \times n$ con numeri reali e non-negativi sulla diagonale, questi numeri sono detti *valori singolari* di M e sono univocamente determinati;
- V^* è la trasposta coniugata di una matrice unitaria $n \times n$ le cui colonne sono autovettori di $M^* \cdot M$.

A differenza della scomposizione spettrale, che non è sempre definita, la SVD è definita per tutti i tipi di matrici appartenenti a $M^{m \times n}(\mathbb{C})$. Nel nostro caso particolare, la matrice che vogliamo fattorizzare è quadrata e simmetrica, quindi i valori singolari sono identici agli autovalori [5].

Se non si ha interesse ad una rappresentazione esatta della matrice M ma ci si accontenta di una sua approssimazione si può usare la cosiddetta *SVD troncata* che prevede di calcolare solo le prime k colonne di U e le prime k righe di V^*. La matrice

$$M_k = U_k \cdot L_k \cdot V_k^* \tag{3.2}$$

e la matrice di rango k che meglio approssima M. Nel prossimo capitolo useremo questo metodo per operare una riduzione dimensionale.

Capitolo 4

Torniamo alle memorie

Forti del nostro nuovo armamentario matematico possiamo dare l'assalto alla correlazione che tanto ci ha fatto penare in precedenza. Utilizzando la matrice ottenuta dalla SVD i vecchi esempi possono essere ridotti arbitrariamente ma se pensiamo a come abbiamo effettuato il calcolo delle componenti principali possiamo già stimare che le dimensioni significative siano vicine al numero degli esempi. Per avere una stima più quantitativa è possibile considerare la frazione della varianza globale che si sta considerando attraverso gli autovalori di C. In formula:

$$\frac{\sum_{i=1}^{i_{max}} \lambda_i}{\mathrm{tr(C)}}. \tag{4.1}$$

Il grafico 4.1 da un'idea di quello che succede: con 26 esempi bastano (ovviamente) 26 dimensioni per rappresentare i dati.

Adesso i vettori sono decorrelati ma abbiamo pagato un prezzo: le componenti non sono più binarie (± 1), bensì numeri reali. In questa situazione tutto l'impianto della rete andrebbe modificato per lavorare con neuroni continui e allora, per non allontanarmi troppo dall'obiettivo di questa tesi, ho preferito utilizzare un piccolo *escamotage* e scrivere in codice Gray le nuove coordinate[1]. Riservando 20 bit di codifica per ognuna le dimensioni crescono un po' ($26 \times 20 = 520$) ma abbiamo il vantaggio di poter usare la rete binaria di prima. Lanciando la simulazione con gli esempi puliti si può subito apprezzare una drastica diminuzione degli errori (si raffronti la figura 4.2 con la 2.3), in questo caso sono talmente pochi che non ha senso cominciare un analisi statistica.

Quello che si può fare, invece, è ripercorrere l'analisi fatta nella sezione 2.5 usando come stati iniziali esempi sporcati, tagliati o spostati. Sono possibili due strade: corrompere i dati prima o dopo la decorrelazione. Per il secondo caso i risultati sono visibili nelle figure 4.3; vediamo subito dei netti miglioramenti nel riconoscimento degli esempi sporcati (vengono individuati

[1] Il vantaggio di questa codifica rispetto alla rappresentazione in base due è che la rappresentazione di numeri vicini differisce al massimo per una cifra

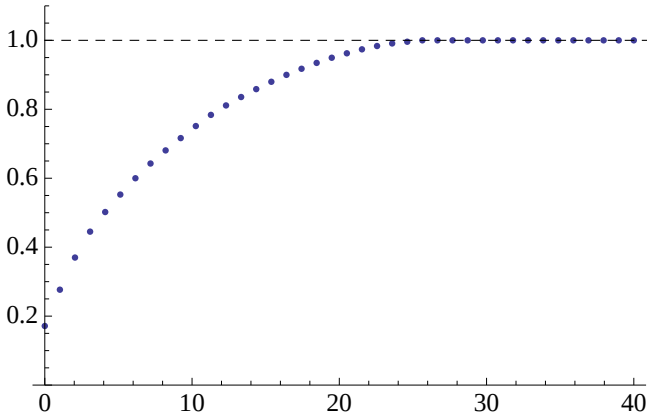

Figura 4.1: Frazione della varianza dei dati in funzione del numero di componenti conservate

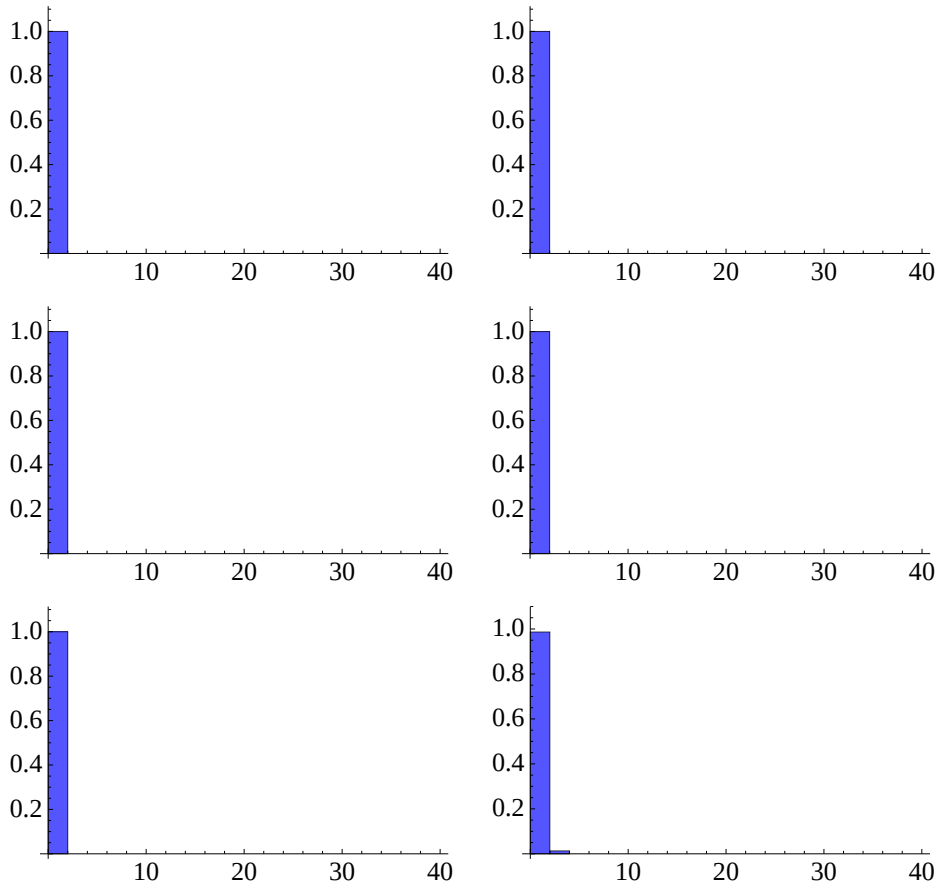

Figura 4.2: I risultati della seconda simulazione con 1, 3, 5, 7, 11 e 15 esempi decorrelati nella memoria.

correttamente fino ad un rumore del'80%) e tagliati (nessun errore anche con mezza immagine sbagliata), mentre nel caso dello spostamento laterale le prestazioni sono leggermente inferiori.

I risultati che lasciano di stucco sono quelli relativi alle immagini corrotte *prima* della decorrelazione (visibili nella figura 4.4): in linea con le immagini non trattate se non peggiori. Bisogna cercare di trovare una spiegazione ragionevole per questo problema, che affligge una delle parti più interessanti di questa analisi. Dopo molto cercare il colpevole è stato individuato nella mappatura bidimensionale delle componenti principali. La scrittura delle componenti selezionate in un nuovo vettore binario, infatti, se da un lato semplifica enormemente la computazione, dall'altro non preserva le distanze nello spazio delle configurazione. Questo comportamento mette in crisi la memoria, che funziona per somiglianza, portando un esempio rovinato vicino ad uno stato diverso da quello di partenza.

Per ovviare a questa situazione serve una rappresentazione binaria delle componenti principali che conservi le distanze, un problema tanto facile a enunciarsi quanto difficile da risolvere. Dopo aver esaminato alcune soluzioni (principalmente [6] e [7]) ho dovuto scartarle in quanto prospettavano sistemi interessanti per trovare delle componenti principali binarie ma senza la possibilità di ottenere nuovi insiemi di dati decorrelati *e* essi stessi in forma binaria.

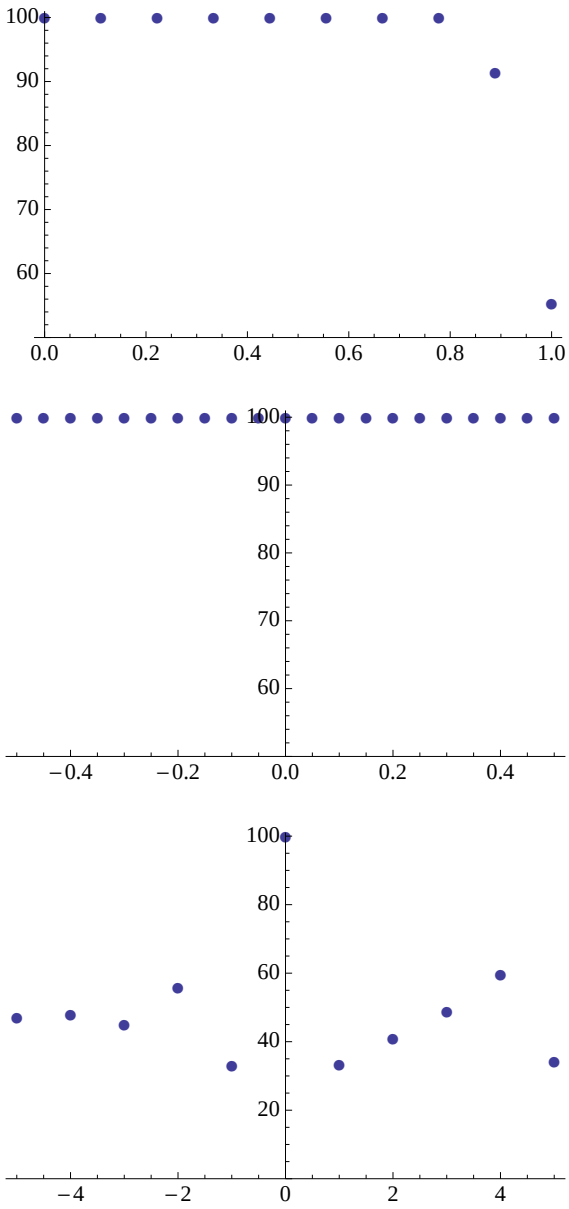

Figura 4.3: Probabilità di recupero per esempi sporcati, tagliati o spostati *dopo* la decorrelazione. La linea rossa nell'ultimo grafico rappresenta l'andamento misurato in precedenza.

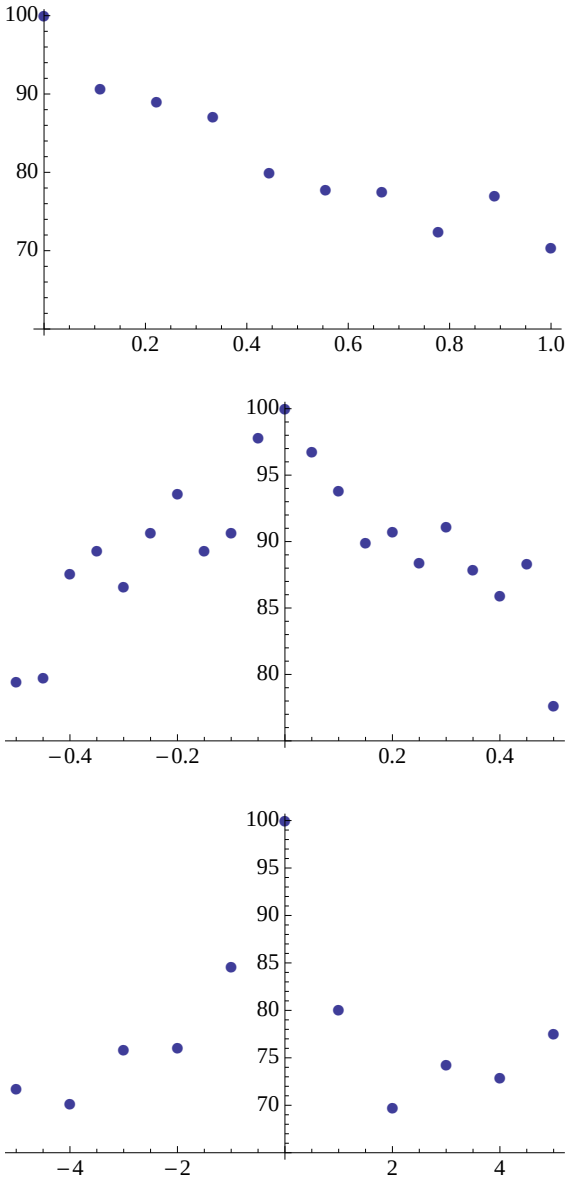

Figura 4.4: Probabilità di recupero per esempi sporcati, tagliati o spostati *prima* della decorrelazione.

Appendice A

L'energia nelle memorie associative

Le funzioni di Ljapunov, battezzate con il nome del matematico russo Aleksandr Michajlovič Ljapunov, servono a provare la stabilità di un punto fisso in un sistema dinamico. L'identificazione di una di queste funzioni semplifica di molto l'analisi di questi sistemi (si pensi all'energia). Vedremo che anche nel caso delle reti di Hopfield è possibile identificare una funzione di questo tipo. Per uno stato s definiamo

$$\mathrm{H}(\mathbf{s}) := -\frac{1}{2}\sum_{ij} s_i s_j W_{ij}; \tag{A.1}$$

dimostreremo che ad ogni aggiornamento dei neuroni corrisponde un $\Delta\,\mathrm{H} < 0$ e che gli esempi memorizzati sono rappresentati dai minimi di $\mathrm{H}(\mathbf{s})$.

Consideriamo che durante l'aggiornamento avvenga $s'_k = -s_k$, il che implica $\operatorname{sgn}(W_k \cdot \mathbf{s}) = -\operatorname{sgn}(s_k)$, la variazione della funzione di Ljapunov vale

$$\Delta\,\mathrm{H}(\mathbf{s}) = -\frac{1}{2}\sum_i s'_k s_i (W_{ik} + W_{ki}) + \frac{1}{2}\sum_i s_k s_i (W_{ik} + W_{ki})$$

e siccome i pesi sono simmetrici possiamo scrivere

$$\Delta\,\mathrm{H}(\mathbf{s}) = 2 s_k \sum_i s_i W_{ik} = 2 s_k W_k \cdot \mathbf{s} < 0\,.$$

Siccome H è una funzione limitata in modulo, questo risultato ci assicura che in un numero finto di iterazioni raggiungeremo un minimo della funzione (non è detto che sia il minimo assoluto, però).

Rimane da verificare che gli esempi memorizzati corrispondano agli stati di minimo. Per farlo sostituiamo ai pesi nell'equazione (A.1) la regola di Hebb (2.3):

$$\mathrm{H}(\mathbf{s}) = -\frac{1}{2n}\sum_{\nu=1}^{m}\sum_{ij} s_i s_j \xi_{\nu i}\xi_{\nu j} = -\frac{1}{2n}\sum_{\nu=1}^{m}\sum_{i}(s_i \xi_{\nu i})^2 = -\frac{1}{2n}\sum_{\nu=1}^{m}(\mathbf{s}\cdot\boldsymbol{\xi}_\nu)^2$$

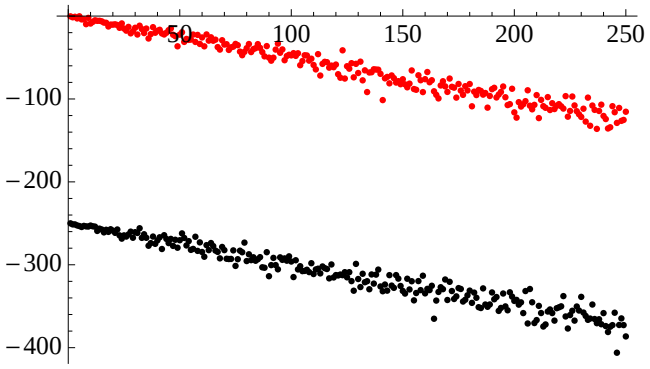

Figura A.1: Valori di H(**s**) per $n = 400$ a m crescenti. I punti neri rappresentano un **s** scelto fra gli esempi con cui è stata costruita la matrice dei pesi, i rossi un vettore generato casualmente.

da cui possiamo vedere che se $\mathbf{s} = \pm\boldsymbol{\xi}_\mu$ almeno uno dei termini della sommatoria vale n^2 e nel caso si un solo esempio ritroviamo i risultati precedenti. Per $m > 1$ si può fare una stima usando l'ipotesi di esempi totalmente scorrelati, in questo caso la differenza tra far partire la rete in uno degli esempi o in uno stato a caso è mostrata nella figura A.1.

Ringraziamenti

Grazie a tutti.

Bibliografia

[1] D. Amit, H. Gutfreund, and H. Sompolinsky. Statistical mechanics of neural networks near saturation. *Annals of Physics*, 175:30–67, 1987.

[2] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley, 1991.

[3] J. J. Hopfield. Hopfield network. *Scholarpedia*, 2(5):1977, 2007.

[4] W. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 7:115–133, 1943.

[5] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes: The Art of Scientific Computing*, chapter 11 - Eigensystem, pages 569–570. Cambridge University Press, third edition, 2007.

[6] Andrew I. Schein, Lawrence K. Saul, and Lyle H. Ungar. A generalized linear model for principal component analysis of binary data. In *Proceedings of the 9'th International Workshop on Artificial Intelligence and Statistics*, 2003.

[7] Feng Tang and Hai Tao. Binary principal component analysis. In *Proceedings of British Machine Vision Conference*, volume 1, pages 377–386, 2006.