

Introduzione alla Teoria delle Reti Neurali

Marco Budinich

Dipartimento di Fisica e INFN
Via Valerio 2, 34127 Trieste — Italy

`mbh@ts.infn.it` — <http://www.ts.infn.it/~mbh/MBHgeneral.html>

a.a. 2016–2017 — Versione 2.1

Corso 6 CFU: 139SM, SSD INF/01

22 dicembre 2016

1 Introduzione

La mente umana ha affascinato l'uomo sin dall'antichità quando ha iniziato a studiarla con l'unico mezzo allora a disposizione: la filosofia. Dopo moltissimo tempo si è aperta un strada nuova per questo studio: quella battuta dallo studio anatomico e dalla fisiologia. Ancor più di recente si è aperta una terza strada basata sullo studio delle proprietà di reti di neuroni, o meglio di modelli astratti di neuroni: noi seguiremo questa pista ancora fresca.

In questo cammino prenderemo un approccio di basso profilo e, senza mai pretendere di fare affermazioni sulla mente, o più in generale sui sistemi biologici, ci limiteremo rigorosamente a studiare le proprietà matematiche di un modello altamente idealizzato delle reti di neuroni reali. Nel secolo scorso ci sono stati alti e bassi nello studio di questa materia, in particolare l'entusiasmo è salito a valori altissimi in tre periodi distinti: nel 1943 con McCulloch e Pitts, cui si è aggiunto Hebb nel 1949, con i modelli matematici del neurone; nel 1958–62 con Rosenblatt e il Perceptron che era sembrato ad alcuni la soluzione finale (con il brusco ritorno alla realtà dovuto a Minsky e Papert nel 1969) e infine nel 1980 con Hopfield ed il gruppo PDP (Parallel Distributed Processing) e la nascita della back-propagation; avremo modo di parlarne.

Ci sono compiti che risultano banali per noi ma al contrario sono difficilissimi da programmare su un computer tradizionale tipicamente sono i problemi di 'pattern recognition' il riconoscimento di forme, strutture, relazioni in insiemi di dati rumorosi come per esempio nella visione, nel riconoscimento della voce etc.. Tradizionalmente le reti neurali si trovano applicate a problemi di questo tipo su ispirazione dei sistemi sensoriali e nella speranza di riuscire a superare le difficoltà incontrate dai computer tradizionali. Questo ha fatto sì che storicamente il punto di vista adottato dalle reti neurali sia completamente diverso da quello dei computer Von Neumann dove si parte invece dagli algoritmi e dalla loro esecuzione efficiente. Nondimeno troveremo che anche partendo dalla pattern recognition alla fine si i due cammini si ricongiungono e arriveremo nuovamente agli algoritmi anche se li troveremo formulati in modo diverso. Solo molto di recente sembra che alcune reti neurali dedicate (per esempio al riconoscimento della voce umana, vedi su: https://en.wikipedia.org/wiki/Deep_learning) siano diventate competitive con algoritmi più tradizionali.

2 Una breve introduzione biologica

Questa parte è tratta dai libri di Griffith [6] e Müller e Reinhardt [10], vedi anche su: http://en.wikipedia.org/wiki/Action_potential.

Il cervello dei mammiferi è un sistema superiore ad un moderno computer sotto molti punti di vista:

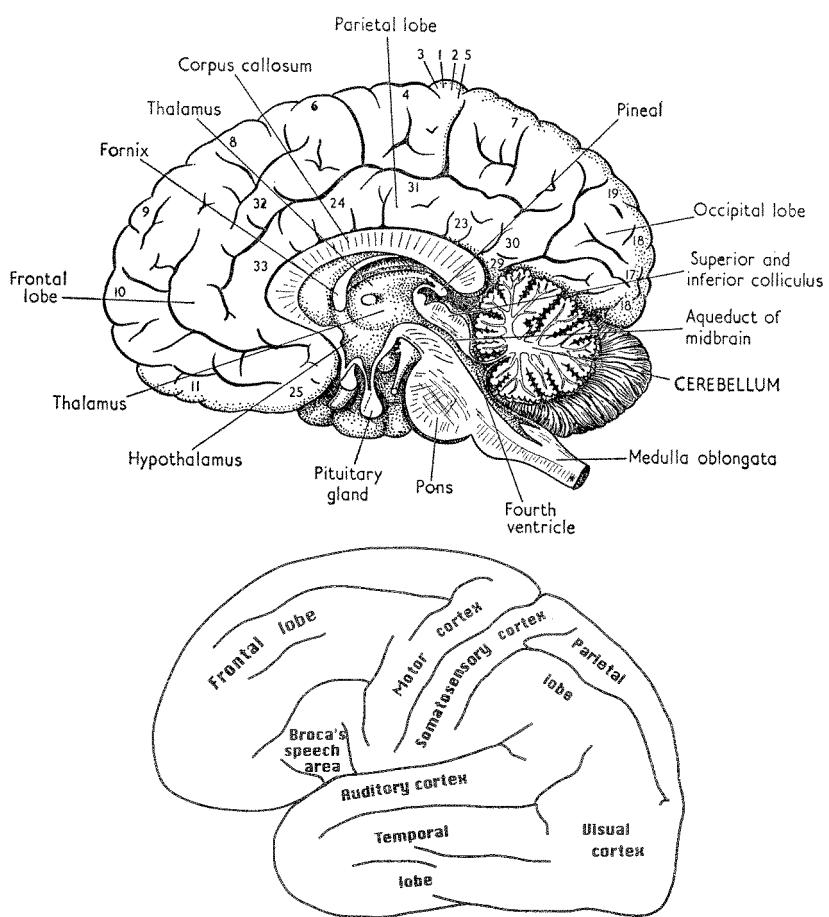


Figura 1: Sezione verticale di cervello umano e mappa delle aree specializzate, vedi anche la figura 32.

- è robusto e resistente ai guasti: la morte di uno o più cellule nervose non gli impedisce di funzionare,
- è flessibile: con l'apprendimento si adatta facilmente a nuove condizioni,
- è efficiente: quello umano ha un consumo energetico compreso fra 1/2 e 25 W (dipende dall'attività),
- è intrinsecamente e massicciamente parallelo,
- può trattare facilmente informazione incompleta o inconsistente.

Il cervello umano pesa in media 1400 grammi (ma quello di Bismarck ne pesava 1807 !) e la maggior parte della massa è concentrata negli emisferi cerebrali che sono formati dalla corteccia. Essa è una superficie di 2–3 mm di spessore fortemente ripiegata in convoluzioni. La corteccia è divisa in aree specializzate come ad esempio la corteccia visiva, quella motoria etc., vedi figura 1. Queste aree sono divise in “microcolonne”, disposte ortogonalmente rispetto alla superficie della corteccia e dedicate a elaborare compiti ancor più specializzati. Ogni microcolonna è ricchissima di connessioni interne, vedi figura 2, mentre è molto meno connessa con le microcolonne adiacenti. Si può dire che lo studio del cervello inizia con Camillo Golgi (1880) che scopre un metodo per colorare le cellule nervose rendendole visibili al microscopio e da Santiago Ramon y Cajal (1880) che divide con lui il premio Nobel per la medicina nel 1906 per la scoperta che il sistema nervoso è fatto di cellule nervose separate: i *neuroni*.

Per quanto ne sappiamo il cervello umano è anche la struttura più complicata dell'universo: è formato all'incirca da 10^{11} neuroni ognuno dei quali è a sua volta connesso con 10^3 neuroni per un totale approssimativo di connessioni che è dell'ordine di 10^{14} . Il tempo tipico caratteristico di un neurone è dell'ordine dei 10^{-3} secondi.

Anatomicamente i neuroni sono di moltissimi tipi ma si possono suddividere in neuroni sensoriali, che ricevono informazione dall'esterno, neuroni motori che controllano i muscoli e neuroni “interneurali” connessi solo ad altri neuroni che sono la maggioranza. Dopo lo sviluppo fetale i neuroni non si moltiplicano più però possono morire. Oltre ai neuroni ci sono le cellule gliali che riempiono gli spazi fra neurone e neurone, esse sono deputate al trasporto dei metaboliti e possono moltiplicarsi. Pur essendo di vari tipi i neuroni hanno in comune alcune caratteristiche:

- il soma che è il corpo principale della cellula neurone,
- i dendriti che sono ramificazioni che portano i segnali elettrici *al neurone*,

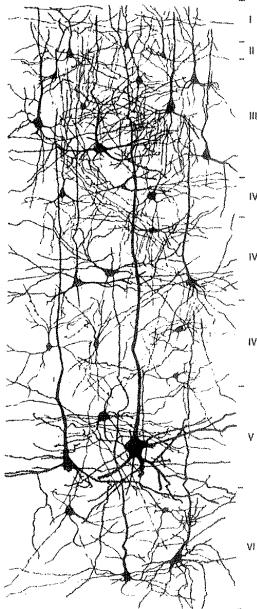


Figura 2: Sezione di una microcolonna di neocorteccia visiva (spessore in verticale) nella quale sono colorati, e dunque visibili, solo circa l'1% dei neuroni, si veda anche in figura 69.

- l'assone che è un sottile tubicino, che può essere lungo da alcuni μm ad oltre un metro, e trasporta il segnale elettrico generato *dal neurone* e termina con delle ramificazioni, la velocità di propagazione del segnale è compresa fra 0.5 e 2 m/sec. Alcuni neuroni, specialmente quelli sensoriali con un lungo assone, hanno l'assone ricoperto da un materiale isolante, la guaina mielinica, che rende più veloce la trasmissione del segnale elettrico lungo l'assone (un pò come in una guida d'onda) e la velocità può arrivare a 100 m/sec, vedi in figura 3.

Se si misura la differenza di potenziale fra l'interno e l'esterno della membrana cellulare di un neurone a riposo si osserva che è $V = -70$ mV; il che significa che la membrana, spessa circa 70 \AA ($70 \times 10^{-10} \text{ m}$), è attraversata da un campo elettrico di ben 10^7 V/m ! La membrana dei neuroni è una struttura molto complicata, selettivamente permeabile agli ioni K^+ Na^+ e Cl^- ; il suo funzionamento è ben capito fenomenologicamente ma, a livello molecolare, presenta ancora dei lati oscuri. La permeabilità selettiva della membrana fa sì che le concentrazioni dei vari ioni siano quelle esposte in tabella 1, in pratica K^+ all'interno e Na^+ e Cl^- all'esterno. A questi gradienti di concentrazioni corrisponderebbero dei potenziali di equilibrio V_K , V_{Na} e V_{Cl} , dati dall'equazione di Nernst, che è quella che appare nell'ultima riga di tabella 1. Visto che la differenza di potenziale osservata a riposo è invece $V = -70$ mV si deduce che, mentre il Cloro è praticamente in equili-

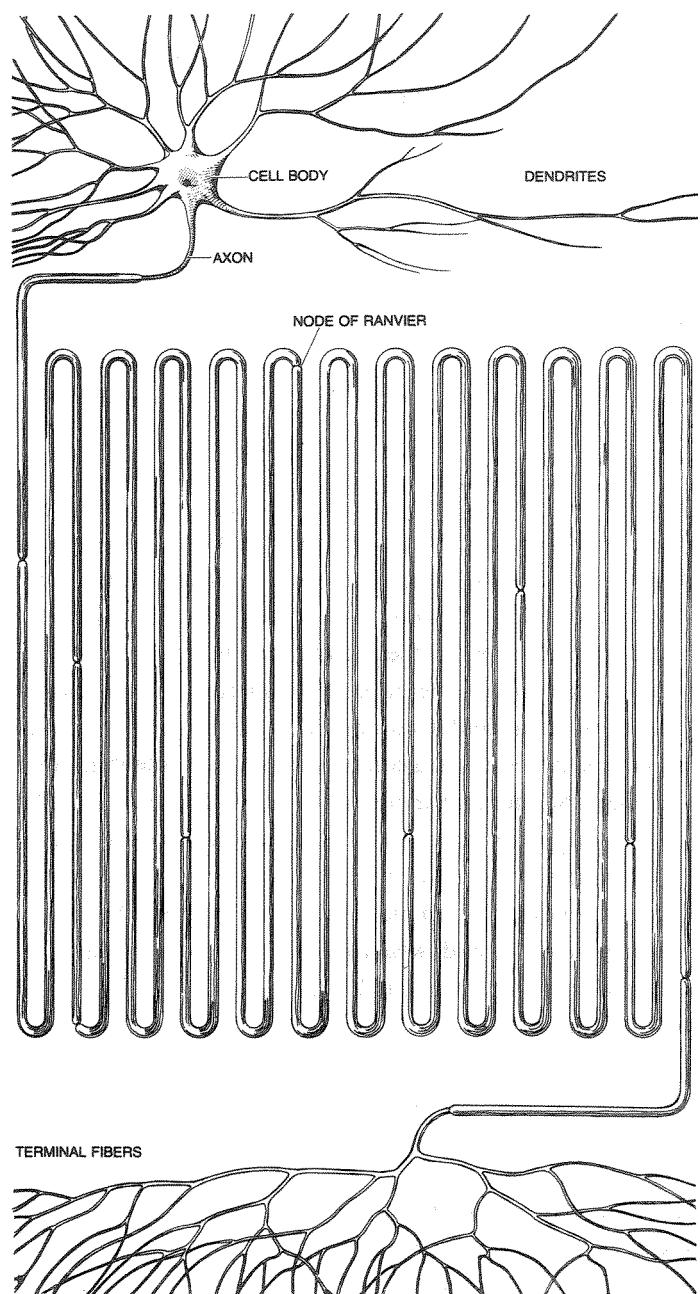


Figura 3: Tipico neurone di vertebrato ingrandito 250 volte; l'impulso nervoso nasce nella cellula e si propaga lungo l'assone che qui appare ripiegato per farlo stare nella figura. In questo caso l'assone è ricoperto dalla guaina mielinica interrotta nei "nodi di Ranvier". Da C.F.Stevens, The Neuron, in The Brain, Scientific American, 1979.

	K^+	Na^+	Cl^-
Esterno (mM)	5.5	150	125
Interno (mM)	150	15	9
V_x (mV)	-90	+60	-70

Tabella 1: Le concentrazioni degli ioni all’interno e all’esterno della membrana del neurone e relativi potenziali di equilibrio.

brio, le concentrazioni di Potassio e Sodio sono in uno stato stazionario che necessita di un *trasporto attivo* da parte della membrana cellulare; questo trasporto attivo consuma energia.

Esiste un modello fenomenologico del comportamento elettrico della membrana, proposto da Hodgkin e Huxley nel 1952 (premio Nobel), che schematizza un pezzetto di membrana con il circuito riprodotto in figura 4.

Si può calcolare la corrente che attraversa la membrana come la somma di tre termini (trascorrendo contributi da altri ioni)

$$i = C \frac{dV}{dt} + \frac{V - V_{Na}}{R_{Na}} + \frac{V - V_K}{R_K}$$

e all’equilibrio, quando $i = \frac{dV}{dt} = 0$, facilmente si ricava per la differenza di potenziale V fra interno ed esterno

$$V = \frac{R_K V_{Na} + R_{Na} V_K}{R_{Na} + R_K}$$

cioè una media pesata fra V_K e V_{Na} dal che si deduce che $V_K < V < V_{Na}$.

Se il potenziale V viene modificato, per esempio per l’arrivo di segnali elettrici dai dendriti, accade che i valori di R_{Na} e R_K cambiano bruscamente, o almeno questa è l’interpretazione in questo modello (in realtà nel modello questi valori dipendono anche dai valori passati di V). Se V supera un valore di soglia V_0 che si aggira intorno ai -50 mV allora R_{Na} tende rapidamente a 0, cioè la membrana diventa permeabile agli ioni Na^+ , che entrano violentemente e in una frazione di msec il potenziale sale fino a raggiungere V_{Na} . Contemporaneamente, più lentamente, anche R_K diminuisce per cui il potenziale ritorna successivamente a V_K . Questo processo, che si svolge tutto in circa 1 msec, viene descritto dicendo che il neurone ha “sparato” (firing in inglese), vedi figura 5. Una volta ritornato a V_K i valori delle resistenze ritornano a salire mantenendo il rapporto costante e, passato un periodo di refrattarietà di circa 1 msec durante il quale il neurone non può sparare, si ritorna nella condizione di riposo, V_r in figura 5 (in realtà c’è anche un periodo di refrattarietà secondaria, di durata maggiore, nel quale il neurone ha minor sensibilità agli stimoli).

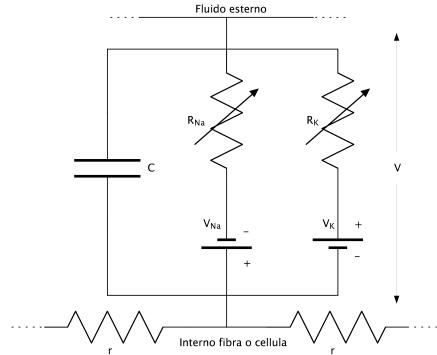


Figura 4: Circuito elettrico equivalente alla membrana neuronale.

Figura 4: Circuito elettrico equivalente alla membrana neuronale.

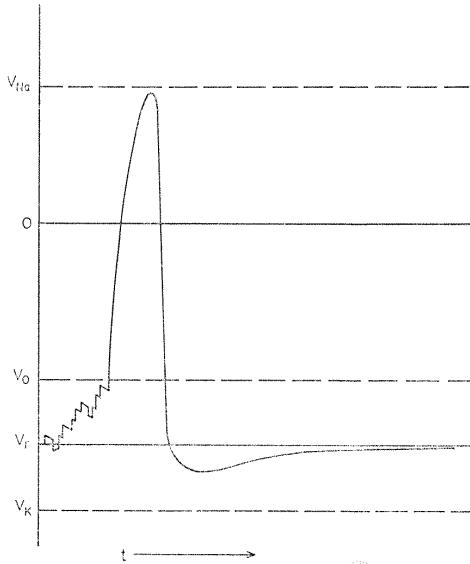


Figura 5: Andamento di V in funzione del tempo per un neurone che spara, lo sparo dura in tutto circa 1 millisecondo, V_r indica il potenziale a riposo e V_0 il valore del potenziale di soglia.

Abbiamo esaminato il fenomeno di sparo per un circuito che simulava un pezzetto di membrana. Se immaginiamo la membrana costituita da tanti di questi circuiti, uno vicino all’altro, si capisce facilmente che una volta che un circuito ha sparato, attraverso la resistenza r interna al neurone, si alza il potenziale dei circuiti adiacenti fino a superare il valore di soglia e anche per essi inizia il processo. In questo modo, in una specie di effetto domino, lo sparo si propaga su tutta la membrana cellulare ed in particolare lungo tutto l’assone e le sue ramificazioni finali.

Queste ramificazioni terminano con delle sinapsi che contengono vescicole di neurotrasmettitori che vengono rilasciate quando arriva lo sparo. In questo modo lo sparo di un neurone genera un segnale chimico alle sinapsi il cui effetto è di variare il potenziale elettrico del neurone successivo, cioè quello proprietario dei dendriti. Se quest’ultimo riceve sufficienti segnali questi possono far alzare il suo potenziale fino ad oltre il livello di soglia portando anche questo neurone a sparare. In questo modo avviene la trasmissione di segnali da un neurone all’altro.

Le sinapsi possono rilasciare diversi tipi di neurotrasmettitori che hanno l’effetto di alzare (eccitare) od abbassare (inibire) localmente il potenziale del neurone destinatario¹. Inoltre le sinapsi possono contenere più o meno vescicole di neurotrasmettitore in modo che l’effetto può essere più o meno

¹Neurotrasmettitori eccitatori sono per esempio l’acido glutammico e l’acetilcolina, inibitori l’acido gamma-amminobutirrico (GABA) e la glicina.

intenso. Le variazioni di potenziale dovute ai singoli rilasci di vescicole sono visibili nella prima parte della curva di figura 5, quando l'effetto dei singoli rilasci fa sì che il potenziale raggiunga il valore di soglia il neurone spara. Dopo il rilascio i neurotrasmettitori vengono rapidamente (circa 1 msec) neutralizzati da enzimi presenti nelle vicinanze per cui lo stimolo cessa rapidamente (per inciso la cocaina interferisce con questi enzimi impedendo ai neurotrasmettitori rilasciati di essere neutralizzati).

L'informazione trasmessa dai neuroni è dunque in forma di segnali elettrici (spari) che possono avere una frequenza che tipicamente va da 1 a 100 Hz. Per esempio nei neuroni sensoriali a maggiore intensità di sensazione corrisponde maggiore frequenza di sparo dei neuroni relativi. Oggi non è ancora chiaro se vi sia contenuta informazione anche nel tempo preciso di sparo del neurone.

3 Modelli matematici ispirati ai neuroni

Una prima pietra miliare dello studio delle reti neurali viene dal lavoro di [McCulloch](#) e [Pitts](#), vedi figura 6, che nel 1943 propongono un modello matematico semplificato del neurone (un po' come Hodgkin e Huxley che qualche anno dopo proporranno un modello elettrico semplificato della membrana). Il loro neurone matematico è basato su questi ingredienti:

- il neurone, in figura 7, ha n ingressi ed un uscita, tutti digitali per rappresentare i due stati dei neuroni: a riposo e in sparo,
- gli ingressi x_i entrano nel neurone con pesi w_i che rappresentano le diverse intensità di contatto sinaptico,
- all'interno del neurone viene calcolata la somma pesata degli ingressi, chiamata *attivazione*, a somiglianza del neurone reale dove la differenza di potenziale totale è la somma delle differenze di potenziale generate dalle singole sinapsi attivate,
- se l'attivazione raggiunge o supera un valore di soglia θ il neurone spara.



Figura 6: Warren McCulloch (1898–1972) e Walter Pitts (1924–1969)

Definita la funzione $\Theta(x)$ che vale 0 per $x < 0$ e 1 per $x \geq 0$ un neurone di McCulloch e Pitts si può definire matematicamente con

$$y = \Theta \left(\sum_{i=1}^n x_i w_i - \theta \right) \quad (1)$$

Inoltre gli autori sono in grado di dimostrare che una rete sincrona di neuroni di questo tipo può calcolare ogni funzione logica, basta partire (è semplice farlo e lo suggerisco come esercizio) da neuroni che implementano le funzioni logiche NOT e AND e, di conseguenza, si può costruire una [macchina di Turing](#) per cui una rete di neuroni di questo tipo può calcolare tutto quanto è programmabile su una macchina di Turing. Vedremo successivamente (capitolo 6) che una rete di questi neuroni può anche approssimare qualsiasi funzione. In sostanza una rete di questi neuroni è assai potente nonostante la forma (1) sia apparentemente semplice.

Quali sono le differenze fra un neurone reale ed uno di McCulloch e Pitts ? Ecco le principali unite a qualche considerazione su come cercare di ridurle ove possibile:

- alcuni tipi di neuroni reali hanno una risposta graduale e non binaria; a questo si può ovviare sostituendo una funzione derivabile al posto di Θ ;

- i neuroni reali non sempre sommano gli ingressi in modo perfettamente lineare: per esempio l'attività su un particolare ingresso può inibire lo sparo del neurone in tutti i casi. Questa non linearità si può riprodurre aggiungendo ulteriori neuroni di McCulloch e Pitts;
- i neuroni reali codificano l'intensità dello stimolo con la frequenza di sparo. Questo è un problema serio e non basta nemmeno introdurre una funzione derivabile al posto di Θ . Inoltre non si può affermare con certezza che non vi sia informazione nel tempo esatto di sparo dei neuroni reali, anche se molti autori non lo ritengono plausibile;
- i neuroni reali non hanno un ritardo fisso né sono perfettamente sincroni caratteristiche invece implicite in una rete di neuroni di McCulloch e Pitts;
- dato che le quantità di neurotrasmettore rilasciato da una sinapsi variano da sparo a sparo c'è una certa casualità nei neuroni reali; a questo problema si può ovviare aggiungendo una certa stocasticità ai neuroni di McCulloch e Pitts.

A questo punto si tratta di capire cosa vogliamo rappresentare con una rete di neuroni: se siamo interessati al funzionamento dettagliato dei neuroni dovremo studiare un sistema in cui si tenga conto di tutte le grandezze reali in gioco come per esempio caratteristiche fisiche della membrana cellulare, pompe sodio potassio, quantità di neurotrasmettitori rilasciati etc. etc. Chiaramente si

tratta di uno studio estremamente complesso ma, se riusciamo a capire il funzionamento di tutte le sue parti, riusciremo a modellare e prevedere il funzionamento di reti di neuroni reali. Questo approccio viene attivamente perseguito soprattutto da chi è interessato alla fisiologia e riesce a descrivere in maniera soddisfacente sistemi formati da pochi neuroni, per esempio si riescono a modellare animali molto semplici che hanno il sistema nervoso fatto da uno o pochi neuroni (uno dei più studiati è l'Aplysia, vedi su <http://en.wikipedia.org/wiki/Aplysia>). Già animali più complessi, come una comune mosca, non possono essere analizzati con un approccio di questo tipo.

Se però siamo disposti a rinunciare alla somiglianza biologica potremo studiare il funzionamento di reti di neuroni di McCulloch e Pitts con la spe-

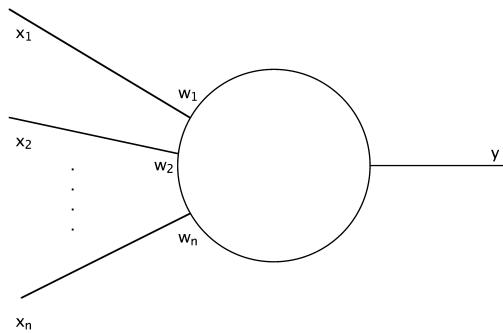


Figura 7: Il modello di neurone di McCulloch e Pitts.

ranza che riescano a rappresentare in maniera soddisfacente alcune caratteristiche dei sistemi biologici al livello più elevato, cioè al livello di algoritmi e memorizzazione dei dati. Come è intuitibile noi prenderemo questa seconda strada, battuta da parecchi anni da coloro che si occupano di reti neurali.

L'assunto, spesso non espresso esplicitamente, è che una rete di McCulloch e Pitts sia in grado di riprodurre le proprietà che consideriamo più salienti di una rete di neuroni reali, cioè le capacità di trattare l'informazione. Come vedremo questo assunto è almeno in parte giustificato a posteriori da alcune caratteristiche interessanti che si possono ritrovare nelle reti di questo tipo.

Possiamo cercare di chiarire questo punto con un esempio un po' paradossale: supponiamo di essere nei panni di un extraterrestre evoluto che si trovasse a dover esaminare un giorno uno dei nostri satelliti più recenti. Si accorgerebbe ovviamente di trovarsi di fronte ad un sistema complesso e rapidamente capirebbe che le funzioni "intelligenti" sono tutte concentrate in una serie di pezzettini di silicio collegati in modo molto complesso da sottilissimi fili di metallo. Egli (ella ?) cercherebbe di approfondire il funzionamento di questi sistemi ma si troverebbe di fronte a una difficoltà: se cominciasse a studiare nel dettaglio il silicio si metterebbe su una strada molto difficile (un moderno chip di cpu o memoria è *estremamente* complicato se analizzato a livello microscopico) per arrivare dopo lunghi studi a capire che esso realizza un'architettura piuttosto semplice (il modello Von Neumann che, volendo, si può riprodurre con un semplice calcolatore meccanico: la [macchina di Turing](#)). Arrivato a questo punto si troverebbe ancora integralmente davanti il problema di analizzare il software, che in molti casi fa riferimento a strutture mentali ancor più complesse: basti ricordare che sono almeno 2.000 anni che inventiamo e perfezioniamo algoritmi. La conclusione che ne traiamo è che lo studio dell'hardware è importante ma non meno importante è lo studio degli algoritmi che vi sono eseguiti e per capire i quali non necessariamente occorre aver capito il funzionamento dei chip nei più minimi dettagli.

Per tornare al nostro caso pensiamo che studiando le reti di McCulloch e Pitts assumiamo che le grandi complessità fisico-biologiche dei neuroni siano marginali per gli algoritmi che implementano, per cui decidiamo di concentrare gli sforzi verso la comprensione degli algoritmi e dei modelli sui quali sono basati.

Questo dualismo fra algoritmo e supporto materiale sul quale viene implementato è molto generale: possiamo facilmente trovare semplici algoritmi che possano venire eseguiti indifferentemente su diversi hardware per esempio: computer al silicio, computer a valvole, computer meccanici (macchina di Turing), sistemi analogici, reti di neuroni di McCulloch e Pitts, reti di neuroni biologici, etc. etc. Di volta in volta lo stesso algoritmo sarà codificato (programmato) in modo diverso ma il risultato sarà lo stesso su tutti i

supporti, il che dimostra che il supporto *non è essenziale*².

Tutti questi sistemi eseguendo lo stesso algoritmo arrivano agli stessi risultati, naturalmente in tempi diversi e con diverso dispendio energetico. Possiamo dire che tutti hanno dovuto produrre, per arrivare al risultato, la stessa quantità di qualcosa che chiamiamo *calcolo*. Il calcolo si rivela una quantità piuttosto elusiva e difficile da definire esattamente e ragionando su questo argomento ci possiamo porre molte domande interessanti: come si definisce il calcolo ? Si può misurare ? Quali calcoli possono essere fatti su quali hardware ? Quanto calcola un sistema biologico confrontato con un computer ? Si può definire un calcolo minimo ? Corrisponde all'operazione elementare di una macchina di Turing ? Qual è la minima energia necessaria per fare un certo calcolo ?

Sono domande molto interessanti e oggetto di attiva ricerca che noi, purtroppo, non affronteremo.

²Si noti che però si possono anche costruire esempi “opposti”, esempi nei quali il funzionamento dell’hardware è essenziale per la risoluzione del problema: per esempio un computer analogico meccanico costruito con molle che risolve il problema di trovare i coefficienti di una regressione lineare di dati. Questi esempi tuttavia di solito rappresentano soluzioni specifiche di casi particolari.

4 Apprendimento per i neuroni di McCulloch e Pitts

Possiamo pensare al cervello come costituito da 10^{11} oggetti che, in parallelo, eseguono tutti lo stesso calcolo: $y = \Theta(\sum_{i=1}^n x_i w_i - \theta)$, solamente che ognuno ha un “programma” diverso dato dai diversi valori dei pesi w_i . L’ordine dei tempi tipico che ogni neurone impiega per fare il suo calcolo è di 10^{-3} secondi e la grande quantità di calcolo prodotta viene dal grande numero di calcolatori elementari che lavorano in parallelo. Per confronto un normale computer esegue un solo programma ma con tempi tipici dei chip di silicio dell’ordine di 10^{-9} secondi.

Il sogno di tutti coloro che studiano le reti neurali è ovviamente di produrre delle reti di neuroni artificiali che lavorino con i tempi tipici del silicio. Dal punto di vista tecnico questo potrebbe essere realizzato anche oggi ma il punto è che non si sa come programmare una rete di neuroni artificiali; in altre parole, dato un problema, in generale non conosciamo un metodo che permetta di trovare un insieme di pesi con i quali la rete risolva il problema dato.

Nei sistemi biologici è l’apprendimento che produce i pesi dato che quando apprendiamo un compito o memorizziamo dell’informazione di fatto modificiamo i pesi sinaptici (vedi paragrafo 4.3). Per questa ragione molto spesso quando si parla di programmare una rete neurale artificiale si parla di *apprendimento*.

Traendo ispirazione dai sistemi biologici studiamo una forma di apprendimento che consiste nel produrre una serie di esempi e cercare i pesi per i quali, una data rete, impari a riprodurre il risultato che vogliamo. Questo approccio offre un altro vantaggio che è quello di poter fare apprendimento anche solo per esempi senza dover conoscere a priori un algoritmo di soluzione.

Per fare un esempio più specifico supponiamo di voler costruire una rete che riconosca se in un’immagine (in bianco e nero e binaria come un pezzetto di questa pagina) compare una lettera ‘a’³, un compito che ben si adatta alla risposta binaria dei neuroni di McCulloch e Pitts. In questo caso non sarebbe immediato scrivere un algoritmo per un computer tradizionale che risolva il problema ma è invece molto facile produrre una serie di immagini d’esempio e le relative risposte corrette. Per semplificare la cosa supponiamo di voler insegnare questo compito a uno scimpanzè: quello che faremmo sarebbe di preparare una serie di immagini, con le relative risposte, e poi le faremmo vedere al primate premiandolo con una banana per ogni risposta corretta. Ci aspettiamo che in breve tempo la scimmia impari e possa riconoscere correttamente tutte le ‘a’ preparate per gli esempi (e magari anche altre

³tradizionalmente le reti neurali sono applicate a problemi di “pattern recognition” che sono apparentemente facili per noi ma molto difficili da programmare su un calcolatore.

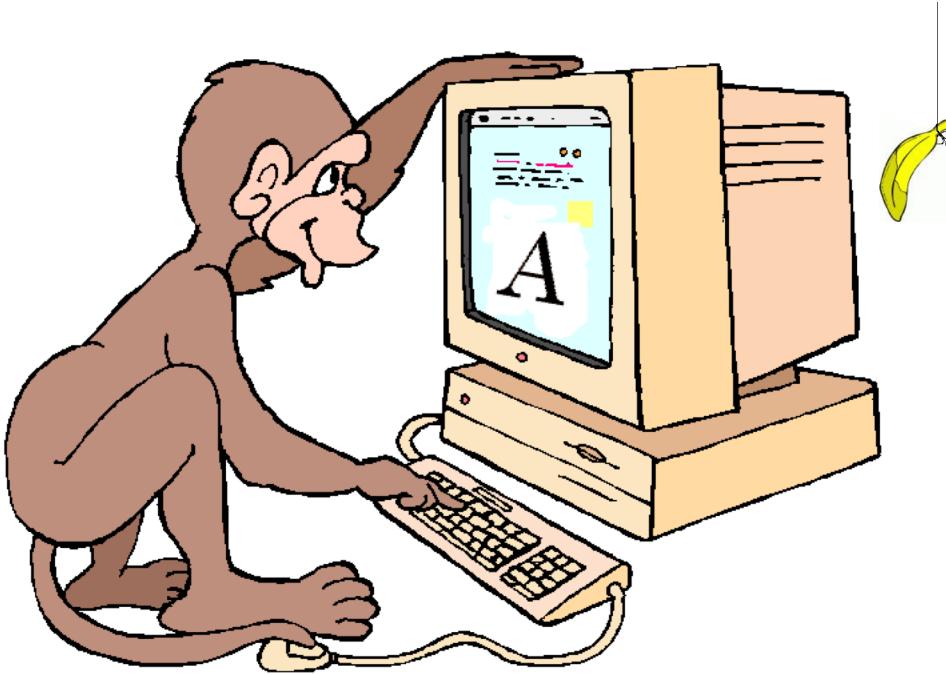


Figura 8: Apprendimento supervisionato.

‘a’). Questo tipo di apprendimento viene detto *supervisionato* dato che c’è un entità, esterna al sistema che apprende, che segnala (con una banana) le risposte corrette (figura 8).

4.1 Formalizzazione matematica e notazione

Trasformiamo ora questo discorso in forma più matematica e introduciamo la notazione che ci accompagnerà per tutto il cammino. Per iniziare supponiamo che la rete cui vogliamo far apprendere questo compito sia fatta da un solo neurone che ha come input i pixel dell’immagine. Indichiamo con \vec{x}_ν una di queste immagini dove il simbolo di vettore indica che l’immagine è fatta da n elementi (pixels) $x_{1\nu}, x_{2\nu}, \dots, x_{n\nu}$ e usiamo l’indice greco ν per indicare un’immagine fra tutte le m immagini di esempio che abbiamo preparato per l’apprendimento. Abbiamo così definito un insieme di esempi

$$\mathcal{E} = \{\vec{x}_\nu : \nu = 1, \dots, m\}$$

a cui associeremo l’insieme delle risposte corrette $\xi_\nu \in \{0, 1\}$ che, nel nostro caso, indica se nell’immagine \vec{x}_ν compare o meno una lettera a. A questo punto supponiamo di avere un neurone di McCulloch e Pitts (1) che riceve in input \vec{x}_ν e in risposta produce l’output $y_\nu \in \{0, 1\}$, in generale diverso da ξ_ν , vedi figura 9.

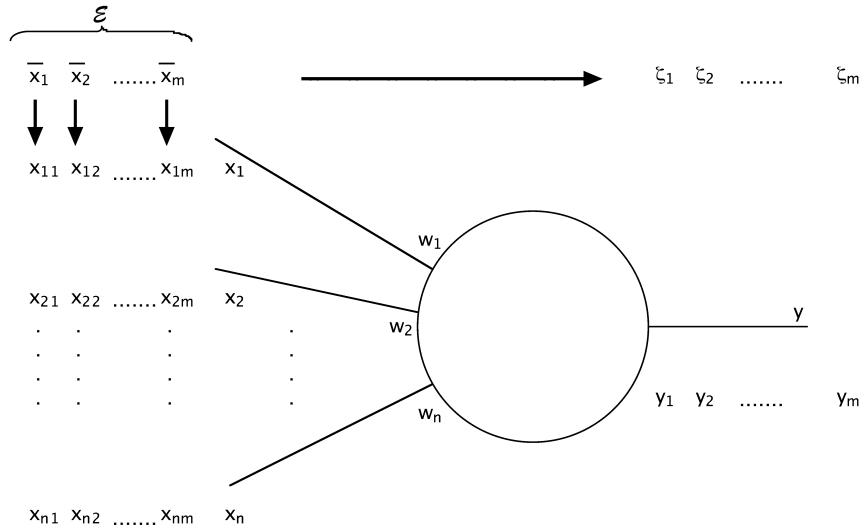


Figura 9: Esemplificazione grafica della notazione introdotta: dando ad un neurone gli m esempi \vec{x}_ν dell'insieme \mathcal{E} in successione, si ottengono le m risposte y_ν , in generale diverse da quelle corrette ξ_ν .

Ora possiamo definire esattamente un *problema*: è l'insieme degli esempi con l'insieme delle relative risposte corrette. Possiamo pensare anche al *problema* come alla tabella di verità che definisce una particolare funzione binaria $f : \{0, 1\}^n \rightarrow \{0, 1\}$. La *soluzione* del problema è data da uno (o più) neuroni che implementano questa funzione, cioè per i quali $y_\nu = \xi_\nu \forall \nu$.

Prima di procedere semplifichiamo la notazione della (1) osservando che se immaginiamo di aggiungere al neurone un ulteriore input x_{n+1} che valga sempre 1 e il cui peso sinaptico sia $-\theta$ la (1) si può riscrivere

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

e in questo modo trattiamo la soglia alla stregua degli altri pesi. Questa somma si può vedere come un prodotto scalare fra due vettori \vec{x} e \vec{w} , per cui possiamo riscriverla in notazione vettoriale:

$$y = \Theta (\vec{x} \cdot \vec{w})$$

dove \vec{x} e \vec{w} sono vettori dello spazio \mathbb{R}^n e dove abbiamo *ridefinito* n come il numero totale dei pesi del neurone, compresa dunque la soglia; in questo caso gli input 'veri' sono $n - 1$ ⁴. In questa notazione possiamo rappresentare un problema come la ricerca di un vettore di pesi \vec{w} tale che:

$$\Theta (\vec{w} \cdot \vec{x}_\nu) = \xi_\nu \quad \nu = 1, \dots, m$$

⁴Si raccomanda di fare attenzione perché alcuni autori usano la convenzione opposta e, come nella (1), ci sono n input 'veri' e $n + 1$ pesi.

Possiamo alleggerire ulteriormente la notazione usando d'ora in poi per tutti i valori binari i valori $\{-1, 1\}$ al posto di quelli $\{0, 1\}$ per cui alla funzione $\Theta(x)$ sostituiremo la funzione segno

$$\text{sgn}(x) = 2\Theta(x) - 1$$

e rappresenteremo matematicamente il nostro neurone con:

$$y = \text{sgn}(\vec{x} \cdot \vec{w}) \quad (2)$$

Questa ridefinizione dello stato di un neurone non è completamente neutrale dato che, come vedremo nel paragrafo 4.3, produce delle differenze nell'apprendimento⁵.

Notiamo che oltre alla stringatezza dalla (2) abbiamo ottenuto una rappresentazione geometrica del problema: infatti possiamo immaginare i vettori \vec{x} e \vec{w} come punti in uno spazio a molte dimensioni e il loro prodotto scalare un'indicazione della loro similarità. Più precisamente siccome \vec{w} rappresenta un iper piano in \mathbb{R}^n , quello di equazione

$$\vec{x} \cdot \vec{w} = 0$$

che passa per l'origine, la (2) può essere interpretata come un *classificatore* che assegna i valori 0 o 1 ai punti \vec{x} a seconda che giacciono da una parte o dall'altra dell'iper piano.

Per aiutare l'intuizione è utile rappresentare gli esempi come punti nello spazio degli input ma fare disegni è possibile solo se lo spazio ha 2 o al più 3 dimensioni. Supponiamo perciò di avere un neurone con 2 ingressi come in figura 11: il suo spazio di input è \mathbb{R}^3 (ricordiamoci che c'è anche la soglia) e la superficie di separazione è un piano di equazione $\vec{x} \cdot \vec{w} = 0$. Sia, per esempio, il vettore di pesi $\vec{w} = (1, 1, 1)$: l'equazione del piano separante è:

$$\vec{x} \cdot \vec{w} = x_1 + x_2 + x_3 = 0$$

e nel disegno a sinistra della figura 10 esso appare in grigio disegnato in un cubo i cui vertici rappresentano le $2^3 = 8$ possibili configurazioni di input. Però la terza coordinata, quella della soglia, è fissa a 1 per cui le uniche configurazioni possibili di input sono solamente $2^{n-1} = 2^2 = 4$ e cioè: $(-1, -1, 1)$, $(-1, 1, 1)$, $(1, -1, 1)$ e $(1, 1, 1)$ e nel disegno sono marcate con dei pallini neri sulla faccia superiore del cubo.

Nel disegno a destra della figura 10 è disegnata solo la faccia superiore del cubo che corrisponde ai 4 possibili input e la linea traccia la separazione che il piano, che passa per l'origine in \mathbb{R}^3 , fa nello spazio \mathbb{R}^2 degli input 'veri'. Notiamo che in questo caso il piano separante non passa per l'origine in \mathbb{R}^2 infatti la sua equazione è $x_1 + x_2 + 1 = 0$ (dato che $x_3 \equiv 1$). Inoltre abbiamo marcato con dei pallini neri tutti gli input che giacciono nel sottospazio per

⁵l'argomento della (1) non cambia effettuando le sostituzioni:

$$\begin{cases} x_i & \rightarrow 2x_i - 1 \\ w_i & \rightarrow w_i/2 \\ \theta & \rightarrow \theta - \sum_{i=1}^n w_i/2 \end{cases}$$

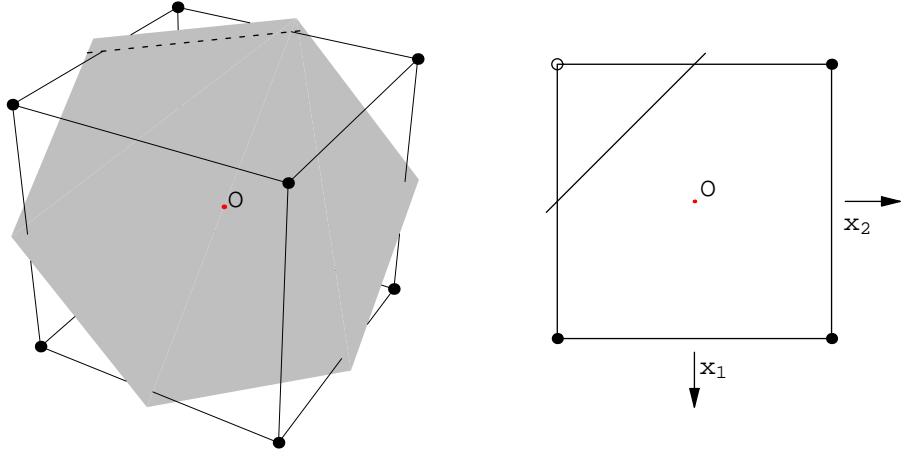


Figura 10: Spazio degli input per un neurone con 2 input 'veri' con e senza la soglia.

il quale l'uscita del neurone è 1 mentre è indicato con un circoletto l'unico input, $(-1, -1)$, per il quale l'uscita del neurone è -1 .

Ne consegue che quando faremo esempi con un neurone a 2 ingressi per semplicità disegneremo solo lo spazio degli input 'veri' e il piano separante non passerà per l'origine. La cosa è ovviamente del tutto generale: ogni iperpiano che non passa per l'origine in uno spazio \mathbb{R}^{n-1} può essere visto come l'intersezione di un iperpiano che passa per l'origine in uno spazio \mathbb{R}^n con l'iperpiano di equazione $x_n = 1$.

Come accennato nel capitolo 3 per rendere più verosimile biologicamente il neurone di McCulloch e Pitts si può introdurre un'altra *funzione di trasferimento* $f(x)$ per cui il caso generale sarà:

$$y = f(\vec{x} \cdot \vec{w}) \quad (3)$$

del quale la (2) diventa il caso particolare in cui $f(x) = \text{sgn}(x)$.

Definiamo infine una funzione di errore E che indica il numero di risposte sbagliate del neurone:

$$E(\vec{w}) = \frac{1}{2} \sum_{\nu=1}^m (\xi_\nu - y_\nu)^2 = \frac{1}{2} \sum_{\nu=1}^m [\xi_\nu - f(\vec{x}_\nu \cdot \vec{w})]^2 \geq 0 \quad (4)$$

è chiaro che è non negativa essendo una somma di quadrati e vale 0 se e solo se tutti i termini della somma sono 0 cioè se la rete è capace di dare

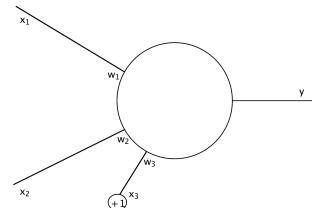


Figura 11: Neurone a 2 ingressi 'veri' (con l'input di soglia fissato a +1).

la risposta corretta per tutti gli m esempi. Questa funzione è definita nello spazio dei pesi \vec{w} dato che al variare dei pesi in generale varia il valore della funzione.

Con l'ausilio della funzione d'errore $E(\vec{w})$ abbiamo trasformato il problema della ricerca dei pesi nella minimizzazione di una funzione. Questa procedura è standard e moltissimi problemi si possono trasformare nella ricerca di un minimo di una opportuna funzione e in particolare nelle reti neurali artificiali questo approccio è molto comune e lo incontreremo più volte in forme leggermente diverse. Nell'appendice A.1 c'è qualche cenno alla minimizzazione di funzioni.

4.2 Altri tipi di apprendimento

L'apprendimento si usa distinguere in due tipi: supervisionato e non supervisionato. Nel primo caso, che abbiamo appena descritto, si suppone che vi sia un entità esterna alla rete che conosce la risposta giusta per un dato esempio \vec{x}_v , e alla quale la rete si rivolge per sapere la risposta corretta. A questo punto una regola di apprendimento permette di tener conto di questa informazione per modificare i pesi se necessario, ad esempio quando la rete ha prodotto una risposta sbagliata. Se la regola di apprendimento è sensata dopo la presentazione di un certo numero di esempi la rete farà meno errori e se non ne fa nessuno diremo che gli esempi sono stati appresi.

Nell'apprendimento non supervisionato invece non c'è nessuna entità esterna e si modificano i pesi con una regola data indipendentemente dal fatto che la risposta prodotta dalla rete sia giusta o sbagliata. Dopo la presentazione di un numero sufficientemente grande di esempi, la rete avrà imparato a classificare gli input, per esempio in base alla somiglianza a input precedenti.

In entrambi i casi si usa l'insieme degli esempi \mathcal{E} sui quali fare l'apprendimento ma sarà poi assai più significativo mettere successivamente alla prova la rete su un diverso insieme di esempi di test \mathcal{T} che non sono mai stati presentati prima alla rete e verificare le sue risposte su questi esempi cioè verificare la cosiddetta capacità di *generalizzazione* della rete. Si vuol verificare se, per caso, la rete non abbia semplicemente memorizzato gli esempi dati senza "capirli".

Notiamo come ci sia una differenza basilare fra programmazione tradizionale e apprendimento: nel primo caso occorre trascrivere un algoritmo in una serie di istruzioni che lo realizzano; nel secondo si stabilisce una regola la cui applicazione fornisce dei pesi che realizzano l'algoritmo di soluzione: potremmo definirlo un *meta-algoritmo*.

4.3 La regola di Hebb



Figura 12: Donald O. Hebb (1904–1985)

Nel 1949 lo psicologo [Donald Hebb](#), figura 12, nel suo libro *Organization of Behaviour* diede tre importanti contributi. Per prima cosa scoprì, sulla base di osservazioni biologiche, che l'intensità dei collegamenti sinaptici può variare nel corso della vita di un neurone: si parla di plasticità sinaptica (per esempio una sinapsi debole può diventare una sinapsi forte e viceversa).

Successivamente associò questa variazione al processo di apprendimento che, ovviamente, modifica il funzionamento dei sistemi di neuroni. Si sa che alla nascita i collegamenti fra certi tipi di neuroni sono pochissimi e crescono nel primo periodo di vita (arborizzazione degli assoni). Questo processo appare inevitabile perché per organismi evoluti non ci sarebbe abbastanza informazione nel DNA per trasmettere una mappa completa dei collegamenti neurali e quindi oltre all'innato occorre un confronto con l'ambiente per uno sviluppo completo.

Infine propose una teoria che prevedeva quando e come avvenisse la modifica delle sinapsi. Questa teoria, nota oggi come la *regola di Hebb*, afferma che, se due neuroni sono sistematicamente attivi allo stesso istante, o più precisamente hanno un'attività correlata, le sinapsi che li collegano vengono rafforzate aumentando così la loro correlazione. Matematicamente dati due neuroni i e j con attività y_i e y_j potremo dire che l'intensità w di una sinapsi che li colleghi venga modificata dalla regola

$$w' = w + \varepsilon y_i y_j$$

dove $y_i y_j$ rappresenta la correlazione fra i due neuroni ed ε è una costante di proporzionalità (nella nostra notazione se y_i entra nell'input k del neurone j scriveremo $\varepsilon x_k y_j$). Vedremo come questa legge, di pura derivazione biologica, sia facile da interpretare anche per i neuroni di McCulloch e Pitts.

Osserviamo che, se applicata ad un neurone di McCulloch e Pitts dove per indicare neuroni quiescenti o in sparo, gli output y assumono valori 0 e 1, con l'apprendimento i valori di w potrebbero diventare solo positivi e rappresentare solo sinapsi eccitatorie. Se invece gli output assumono valori ± 1 le cose sono diverse e ci potrà essere correlazione positiva *anche* fra neuroni che sono entrambi quiescenti e correlazione negativa per neuroni in stati diversi: si parla in questo caso di regola di Hebb *estesa* ed è quella che noi useremo sempre.

Infine notiamo che per aggiornare il valore di una sinapsi c'è solo bisogno di informazione *locale* dato che la sinapsi è modificata solo dall'attività dei neuroni che essa collega: questa risulterà una caratteristica molto saliente.

5 Perceptrons

Iniziamo ad occuparci di un tipo di reti in cui l'informazione va solo in una direzione e non ci sono cicli: le reti *feed-forward* di figura 13; questo tipo di reti sono molto comuni nei sistemi biologici, basti pensare ai sistemi sensoriali, come la retina di figura 64. Esse assomigliano a una funzione matematica $f(x)$ in cui dato un input x si produce l'output corrispondente. Forme più generali di reti, che contengano dei cicli nelle connessioni, sono dette *ricorrenti*, figura 43, sono in generale più complicate e spesso danno origine a fenomeni periodici nel tempo.

Nel 1962 [Frank Rosenblatt](#), anch'egli psicologo, vedi figura 14, pone una seconda pietra miliare nel campo delle reti neurali proponendo il *Perceptron*: un neurone con associata una semplice regola di apprendimento. Rosenblatt dimostra che l'apprendimento avrà sicuramente successo, se ciò è possibile⁶.

Vediamo in dettaglio iniziando a definire un Perceptron come un neurone di Mc Culloch e Pitts con funzione di trasferimento binaria (2), in questo caso il Perceptron agisce come un *classificatore* perché per ogni input produce un bit che tipicamente rappresenta l'appartenenza o meno dell'input a una certa classe. Se, come nell'esempio del capitolo 4, volessimo costruire una retina in grado di riconoscere se nell'immagine è presente una lettera a, sarebbe in linea di principio sufficiente assegnare i pixel digitali della nostra retina, come in figura 15, agli input di un neurone che potrebbe indicare, con il valore della sua uscita per una data immagine, se nell'immagine stessa compare una lettera a. Potremmo poi facilmente avere in parallelo altri 22 neuroni specializzati nel riconoscimento delle altre lettere dell'alfabeto. Questo esempio mostra come, anche con dei semplici classificatori, sarebbe in linea di principio possibile costruire funzioni molto complicate.

Riassumendo con il Perceptron abbiamo un neurone il cui output è dato dalla (2) e gli m esempi \vec{x}_ν cui sono associati gli outputs desiderati ξ_ν e quelli in realtà prodotti y_ν . La funzione $E(\vec{w})$, definita in (4), conta il numero di

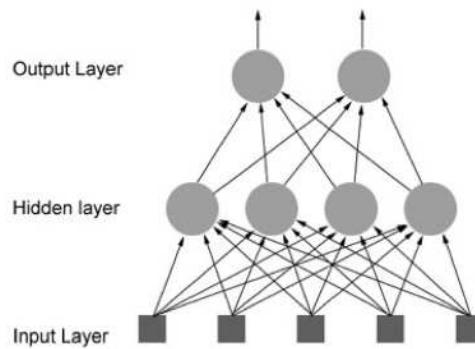


Figura 13: Rappresentazione schematica di una rete feed-forward a tre strati.



Figura 14: Frank Rosenblatt (1928–1971)

⁶pubblica il libro: F. Rosenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*, Spartan Books, 1962.

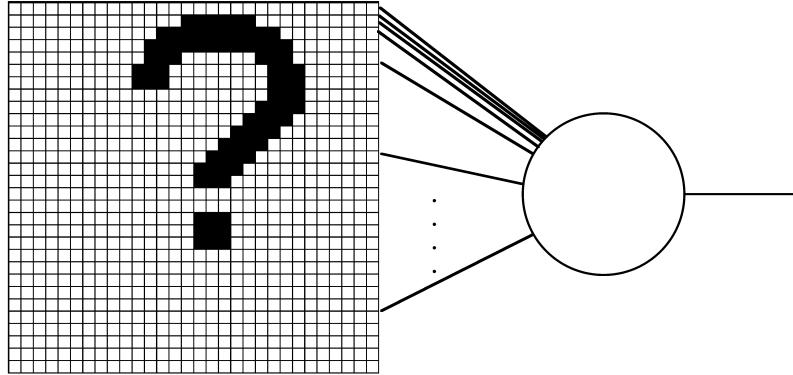


Figura 15: Retina binaria formata da $30 \times 30 = 900$ pixel che sono gli input di un Perceptron.

errori e vale 0 se e solo se, per ogni esempio \vec{x}_ν vale $y_\nu = \xi_\nu$ cioè se

$$\operatorname{sgn}(\vec{x}_\nu \cdot \vec{w}) = \xi_\nu \quad \text{per } \nu = 1, \dots, m$$

che è equivalente alle condizioni

$$\begin{cases} \vec{x}_\nu \cdot \vec{w} \geq 0 & \text{se } \xi_\nu = 1 \\ \vec{x}_\nu \cdot \vec{w} < 0 & \text{se } \xi_\nu = -1 \end{cases}$$

che riassumiamo in una sola relazione moltiplicandole entrambe per ξ_ν

$$\xi_\nu \vec{x}_\nu \cdot \vec{w} \geq 0 \quad \text{per } \nu = 1, \dots, m \tag{5}$$

Questa relazione ci dice che non si commettono errori se il vettore dei pesi \vec{w} ha prodotto scalare non negativo con tutti gli *esempi modificati* $\xi_\nu \vec{x}_\nu$ ⁷.

Secondo la definizione data nel paragrafo 4.1 l'insieme degli esempi \mathcal{E} e quello degli outputs desiderati $\Xi := \{\xi_\nu\}$ definiscono il *problema* la cui *soluzione* consiste nel trovare, se esiste, un vettore dei pesi \vec{w} che soddisfi la (5). L'*apprendimento* invece consiste in una procedura che modifica il vettore dei pesi \vec{w} per cercare di trovare la soluzione.

Osserviamo che la forma (5) ha una chiara interpretazione geometrica: si cerca un vettore \vec{w} che abbia prodotto scalare non negativo con ciascuno degli esempi modificati. Ovverosia si cerca un iperpiano passante per l'origine, che definisce un semispazio che contiene tutti gli esempi modificati.

Nel 1962 Rosenblatt propone una regola di apprendimento, da applicare dopo la presentazione di un esempio \vec{x}_ν , che prescrive di modificare tutti gli n pesi secondo la

$$w_i = w_i + \Delta w_i$$

⁷Si noti che moltiplicando per -1 l'esempio \vec{x}_ν si cambia di segno anche l'input relativo alla soglia per cui gli esempi modificati non giacciono più tutti nell'iperpiano $x_n = 1$.

dove, con questa notazione, intendiamo che il valore di w_i viene modificato sommandoci Δw_i (altre notazioni comuni in letteratura sono $w'_i = w_i + \Delta w_i$ o $w_i \leftarrow w_i + \Delta w_i$) e la variazione del peso è

$$\Delta w_i = \begin{cases} 0 & \text{se } y_\nu = \xi_\nu \\ 2\varepsilon\xi_\nu x_{i\nu} & \text{altrimenti} \end{cases}$$

dove ε è una costante, usualmente piccola ($0 < \varepsilon \ll 1$), che serve a rendere graduale l'apprendimento.

Questa regola è chiaramente ispirata al lavoro di Hebb (paragrafo 4.3) dato che il termine $\xi_\nu x_{i\nu}$ tende ad aumentare quei pesi (sinapsi) per i quali l'input e l'output desiderato sono correlati (uguali) mentre tende a ridurre i pesi per i quali sono scorrelati (opposti).

Possiamo riscrivere la variazione dei pesi in un'unica forma, valida per tutti i casi

$$\Delta w_i = \varepsilon(1 - \xi_\nu y_\nu)\xi_\nu x_{i\nu}$$

dato che la parte in parentesi si annulla quando $y_\nu = \xi_\nu$. Ora che la variazione si può applicare in tutti i casi, possiamo formularla in modo più compatto usando la forma vettoriale

$$\Delta \vec{w} = \varepsilon(1 - \xi_\nu y_\nu)\xi_\nu \vec{x}_\nu$$

per cui la regola di apprendimento completa è

$$\vec{w} = \vec{w} + \varepsilon(1 - \xi_\nu y_\nu)\xi_\nu \vec{x}_\nu \quad (6)$$

che offre anche una semplice interpretazione geometrica: se per un certo esempio il vettore dei pesi \vec{w} produce un risultato sbagliato esso viene modificato aggiungendoci (una frazione 2ε del) l'esempio modificato $\xi_\nu \vec{x}_\nu$, che risultava mal classificato. Visto che si vuole un \vec{w} che abbia prodotto scalare non negativo con tutti gli esempi modificati con questa variazione lo portiamo nella direzione di $\xi_\nu \vec{x}_\nu$, con il quale aveva prodotto scalare negativo. Osserviamo che, se all'inizio dell'apprendimento $\vec{w} = 0$ alla fine, \vec{w} risulterà essere fatto da una pura somma di esempi.

Possiamo riscrivere la regola di apprendimento (6) portando ξ_ν dentro la parentesi e osservando che $\xi_\nu \xi_\nu = 1$

$$\vec{w} = \vec{w} + \varepsilon(\xi_\nu - y_\nu)\vec{x}_\nu \quad (7)$$

che talvolta si trova anche scritta come

$$\vec{w} = \vec{w} + \varepsilon\delta_\nu \vec{x}_\nu \quad (8)$$

dove si indica con δ_ν la differenza fra output desiderato e quello reale.

Siamo ora in grado di dimostrare il risultato principale sui Perceptron che tante aspettative ha generato quando è stato scoperto:

Teorema 1 *Dato un problema per il Perceptron per il quale esista almeno una soluzione \vec{w}^* (tale che $\xi_\nu \vec{x}_\nu \cdot \vec{w}^* \geq 0 \quad \forall \nu$) allora, partendo da $\vec{w} = 0$ e applicando un numero finito di volte la regola di apprendimento (6) si arriva ad un vettore dei pesi che risolve il problema dato.*

Osserviamo che, a parte casi patologici, la soluzione non è unica e quella cui conduce l'algoritmo di apprendimento potrebbe anche essere diversa da \vec{w}^* .

Dimostrazione Supponiamo di aver iniziato l'apprendimento con $\vec{w} = 0$ e di aver applicato R volte la (6) con $\xi_\nu \neq y_\nu$ ed in particolare che l'esempio $\xi_\nu \vec{x}_\nu$ per r_ν volte sia stato classificato erroneamente dunque producendo una modifica del vettore dei pesi. Ovviamente $\sum_{\nu=1}^m r_\nu = R$ e, a questo punto, il vettore dei pesi varrà

$$\vec{w} = 2\varepsilon \sum_{\nu=1}^m r_\nu \xi_\nu \vec{x}_\nu \quad (9)$$

Calcoliamo di quanto varia la lunghezza del vettore dei pesi in seguito ad una singola applicazione della (6). Per brevità estendiamo la notazione $|.|$ ad indicare la lunghezza di un vettore e dunque indicheremo, ove non ambiguo,

$$\vec{w} \cdot \vec{w} := \vec{w}^2 := |\vec{w}|^2$$

Possiamo ora tornare al nostro conto

$$(\vec{w} + 2\varepsilon \xi_\nu \vec{x}_\nu)^2 - \vec{w}^2 = 4\varepsilon^2 (\xi_\nu \vec{x}_\nu)^2 + 4\varepsilon \xi_\nu \vec{x}_\nu \cdot \vec{w} = 4\varepsilon^2 n + 4\varepsilon \xi_\nu \vec{x}_\nu \cdot \vec{w} < 4\varepsilon^2 n$$

dove abbiamo usato la relazione $(\xi_\nu \vec{x}_\nu)^2 = n$ garantita dal fatto che tutte le componenti degli esempi valgono ± 1 . Per l'ultima diseguaglianza abbiamo usato l'informazione che, se è stato usato l'algoritmo di apprendimento, necessariamente valeva $\xi_\nu \vec{x}_\nu \cdot \vec{w} < 0$. Abbiamo dunque un limite superiore all'allungamento del vettore dei pesi in un singolo passo. Necessariamente ne deriva che, dopo R passi di apprendimento, varrà:

$$\vec{w}^2 < 4\varepsilon^2 n R \quad (10)$$

Definiamo la quantità D che, per un dato vettore di pesi \vec{w} , è il minimo valore di $\cos \theta_\nu$ dove l'angolo è quello fra le direzioni di \vec{w} e di $\xi_\nu \vec{x}_\nu$ e cioè:

$$D(\vec{w}) = \min_\nu \cos \theta_\nu = \min_\nu \frac{\xi_\nu \vec{x}_\nu \cdot \vec{w}}{|\xi_\nu \vec{x}_\nu| |\vec{w}|}$$

con l'usuale definizione del prodotto scalare. Dalla (5) deriva facilmente che un problema è risolubile se e solo se esiste un vettore di pesi per il quale $D(\vec{w}^*) \geq 0$ che indica semplicemente che tutti gli esempi modificati hanno prodotto scalare con \vec{w}^* che risulta ≥ 0 . Per evitare inutili complicazioni sui casi patologici dimostreremo il teorema in forma più debole supponendo che la soluzione \vec{w}^* sia tale che $D(\vec{w}^*) > 0$.

Calcoliamo ora il coseno dell'angolo fra il vettore dei pesi trovato con gli R passi di apprendimento (9) e quello \vec{w}^*

$$1 \geq \frac{\vec{w} \cdot \vec{w}^*}{|\vec{w}| |\vec{w}^*|} = 2\epsilon \sum_{\nu=1}^m r_\nu \frac{\xi_\nu \vec{x}_\nu \cdot \vec{w}^*}{|\vec{w}| |\vec{w}^*|} = \frac{2\epsilon}{|\vec{w}|} \sum_{\nu=1}^m r_\nu |\xi_\nu \vec{x}_\nu| \frac{\xi_\nu \vec{x}_\nu \cdot \vec{w}^*}{|\xi_\nu \vec{x}_\nu| |\vec{w}^*|}$$

e messa in questa forma nella somma compare il coseno usato nella definizione di $D(\vec{w}^*)$ per cui facilmente ricaviamo la diseguaglianza

$$\frac{\vec{w} \cdot \vec{w}^*}{|\vec{w}| |\vec{w}^*|} \geq \frac{2\epsilon \sqrt{n}}{|\vec{w}|} \sum_{\nu=1}^m r_\nu D(\vec{w}^*) = \frac{2\epsilon \sqrt{n}}{|\vec{w}|} RD(\vec{w}^*)$$

ed in conclusione abbiamo

$$1 \geq \frac{2\epsilon \sqrt{n}}{|\vec{w}|} RD(\vec{w}^*) .$$

Dato che tutti i termini a destra sono positivi possiamo quadrare la relazione e sostituendo la (10), facilmente ricaviamo

$$1 \geq \frac{4\epsilon^2 n}{|\vec{w}|^2} R^2 D^2(\vec{w}^*) > \frac{4\epsilon^2 n}{4\epsilon^2 n R} R^2 D^2(\vec{w}^*) = RD^2(\vec{w}^*)$$

che possiamo finalmente riscrivere come

$$RD^2(\vec{w}^*) < 1$$

Per ipotesi sappiamo che $D(\vec{w}^*) > 0$ dunque da questa relazione vediamo che il numero totale di passi di apprendimento R non può crescere indefinitamente senza violare la diseguaglianza. Ne dobbiamo dedurre che dopo un certo numero di passi di apprendimento non ci saranno più esempi che richiedano l'applicazione della regola di apprendimento e, di conseguenza, una soluzione del problema sarà stata trovata; il teorema è perciò dimostrato. \square

Possiamo osservare che dall'ultima relazione si ottiene anche un limite massimo al numero di passi di apprendimento necessari per arrivare alla soluzione e cioè:

$$R < \frac{1}{D^2(\vec{w}^*)}$$

che ha la caratteristica, un po' sorprendente, di essere indipendente dal numero degli input n , degli esempi m e da ϵ e di dipendere solo dalla massima distanza fra i vettori degli esempi e il vettore soluzione, cioè da $D(\vec{w}^*)$.

Come già detto la scoperta di questo teorema genera enormi entusiasmi per il Perceptron sui quali però arriva nel 1969 una doccia gelata quando Minsky e Papert pubblicano un libro intitolato "Perceptrons" nel quale dimostrano che, in realtà, la condizione che esista almeno una soluzione \vec{w}^* è

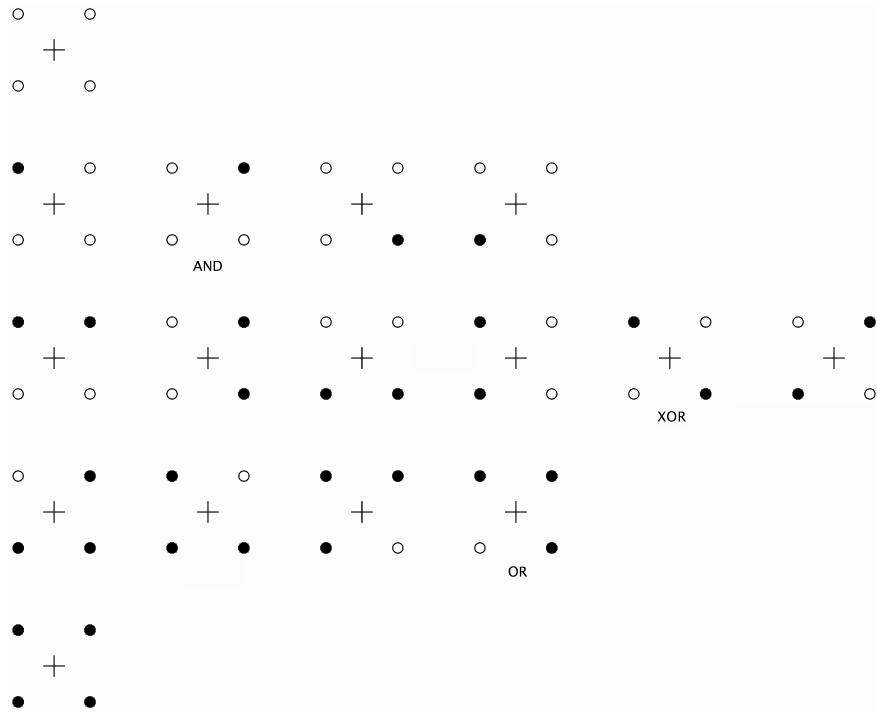


Figura 16: I 16 possibili problemi per un neurone a 2 ingressi.

estremamente restrittiva dato che sono pochissimi i problemi per i quali questa condizione può esser soddisfatta e che, di conseguenza, questo teorema trova applicazione solo per i problemi più semplici. Per esempio si dimostra che il problema di decidere se un certo insieme di pixel neri, su una retina bidimensionale, forma un area connessa o sconnessa, è un problema che in generale non può essere risolto da un Perceptron ed al quale, di conseguenza, il teorema non si può applicare.

Senza affrontare l'esempio delle aree connesse che è più difficile da dimostrare possiamo subito renderci conto delle limitazioni del Perceptron riprendendo il semplice esempio di un neurone con due ingressi che ha dunque 4 possibili configurazioni di input che possiamo rappresentare geometricamente nel piano come in figura 10.

Dato che in questo caso gli input possibili sono solo 4 è ovvio che potremo scegliere i corrispondenti 4 output desiderati $\xi_1 \dots \xi_4$ in $2^4 = 16$ modi possibili; in altre parole per questo Perceptron ci sono 16 problemi diversi che si potrebbero voler imparare. Li possiamo rappresentare graficamente colorando in bianco e nero nei 16 modi possibili i 4 input come si vede in figura 16 ed una soluzione sarà rappresentata da una retta che lasci da una parte tutti i punti neri e dall'altra tutti i punti bianchi.

Non è difficile convincersi che ci sono 14 casi nei quali è possibile trovare una retta che separa i punti bianchi da quelli neri e solo 2 nei quali

questo non è possibile, nella figura 16 questi ultimi sono i due problemi più a destra nella riga centrale. È facile dimostrare analiticamente l'intuizione geometrica ponendo, dal penultimo problema della riga centrale (quello marcato XOR⁸), le richieste sui pesi in forma di un sistema di diseguaglianze

$$\begin{cases} -w_1 - w_2 + w_3 < 0 \\ -w_1 + w_2 + w_3 \geq 0 \\ w_1 - w_2 + w_3 \geq 0 \\ w_1 + w_2 + w_3 < 0 \end{cases}$$

e sommando le 2 diseguaglianze centrali si ottiene $w_3 \geq 0$ mentre sommando la prima e l'ultima si ottiene $w_3 < 0$ da cui si deduce che non esiste un valore di w_3 che possa soddisfare tutte le diseguaglianze e quindi che il problema non ha soluzione.

5.1 E ora ?

Il risultato negativo di Minsky e Papert mette fine bruscamente all'idea che il Perceptron potesse rappresentare la soluzione finale. Per cercare di superare l'ostacolo si seguono tre strade diverse, per ognuna delle quali ci saranno successi e insuccessi. Le vie di procedere sono:

- invece di cercare la soluzione esatta si cerca quella soluzione che rende minimo il numero di errori prodotti dal Perceptron per un dato problema, ne parleremo nel paragrafo 5.2;
- siccome la limitazione discende dal fatto che il Perceptron non può risolvere certi problemi si può cercare di trovare un tipo di rete più potente, ne parleremo nel capitolo 6;
- per capire meglio le limitazioni del Perceptron si può approfondire il tipo e numero di problemi che si possono affrontare, ne parleremo nel capitolo 7.

Anche se nessuno di questi tre approcci risulterà risolutivo nondimeno lungo il cammino incontreremo risultati nuovi e potremo arricchire il quadro generale.

5.2 Il Perceptron continuo

Il primo metodo per cercare di aggirare il fatto che nella maggior parte dei casi il Perceptron non ammette soluzione *esatta* è di rilassare il concetto di soluzione chiedendo di trovare il vettore dei pesi che *minimizza* il numero di errori commessi dal Perceptron. In questo caso si generalizza l'approccio e

⁸Questo problema viene detto XOR (da eXclusive OR) dato che corrisponde alla funzione logica Booleana dell'or esclusivo, interpretando -1 come False e $+1$ come True.

si propone l'apprendimento come la minimizzazione della funzione d'errore $E(\vec{w})$ definita nella (4); se esiste una soluzione esatta troveremo dei pesi che azzerano $E(\vec{w})$, altrimenti si giungerà comunque alla soluzione “meno peggio”.

Un primo beneficio di questa estensione del concetto di apprendimento alla minimizzazione di una funzione d'errore è che risulterà di applicabilità molto più generale infatti lo potremo applicare a tutte le reti che da n input producono un solo output e non al singolo neurone, ne parleremo nel capitolo 6.

L'idea è buona, anche se i risultati non saranno pienamente soddisfacenti, ma per procedere dovremo usare strumenti analitici e una funzione di trasferimento discontinua come $\text{sgn}(x)$ non va bene per cui occorre introdurre un Perceptron continuo che darà una funzione d'errore continua che potremo trattare con i metodi dell'analisi. Il Perceptron si generalizza con l'introduzione di una funzione di trasferimento continua (3) e una scelta assai frequente per $f(x)$ è la tangente iperbolica:

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Questa funzione, riprodotta in figura 17 insieme alla sua derivata, è continua, dispari, limitata fra -1 e $+1$ ed ha il vantaggio che è molto facile calcolarne la derivata se si conosce il valore della funzione dato che⁹

$$\tanh'(x) = 1 - \tanh^2(x)$$

Spesso si aggiunge alla definizione di $f(x)$ un parametro β che moltiplica x (di fatto cambiandone la scala)

$$f_\beta(x) = \tanh(\beta x)$$

e variare β permette di variare la rapidità della transizione in $x = 0$. Per $\beta \rightarrow \infty$ si ha che $\tanh(\beta x) \rightarrow \text{sgn}(x)$ il che ci permette di pensare al Perceptron discreto come a un caso limite di quello continuo.

Anche questa estensione del Perceptron ad una funzione di trasferimento continua porta dei vantaggi:

⁹Se si preferisce una funzione che sia limitata fra 0 e 1 l'altra scelta comune (sempre per le proprietà della derivata) è

$$f(x) = \frac{1}{1 + e^{-2x}}$$

che altro non è che la tangente iperbolica ristretta fra 0 e 1 dato che $2f(x) - 1 = \tanh(x)$. La sua derivata è

$$f'(x) = 2f(x)(1 - f(x))$$

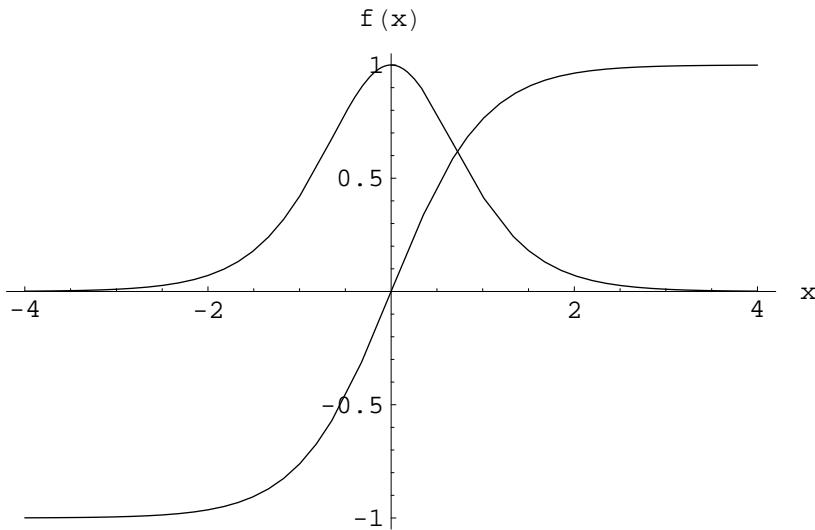


Figura 17: La funzione $\tanh(x)$ e la sua derivata.

- si generalizza la funzione di trasferimento potendo sempre ritrovare quella discreta come caso limite;
- si ottiene una funzione derivabile dunque si apre la porta all'uso degli strumenti analitici;
- si ottiene una maggiore verosimiglianza biologica dato che un output con uscita graduale rappresenta meglio un neurone che spara con frequenza maggiore o minore per indicare l'intensità dello stimolo.

C'è anche uno svantaggio introdotto da questa estensione ed è che in generale risulta più difficile trovare una soluzione esatta dato che se volessimo che l'output, per esempio $y = \tanh(\vec{x} \cdot \vec{w})$, valga esattamente 1 avremmo bisogno di un input infinito alla funzione $\tanh()$.

Per il Perceptron così generalizzato si possono dimostrare proprietà simili a quelle del caso discreto purché si disponga di una buona definizione della regola di apprendimento. Per trovarla ritorniamo alla definizione della funzione d'errore (4)

$$E(\vec{w}) = \frac{1}{2} \sum_{\nu=1}^m (\xi_\nu - y_\nu)^2 = \frac{1}{2} \sum_{\nu=1}^m [\xi_\nu - f(\vec{x}_\nu \cdot \vec{w})]^2$$

essendo essa una somma di quadrati è chiaro che $E \geq 0$ valendo 0 se e solo se tutti gli esempi sono stati appresi perfettamente (a parte il problema di precisione accennato prima). Faremo corrispondere la soluzione del problema alla minimizzazione della funzione $E(\vec{w})$ che però ora sarà continua e derivabile rispetto ai suoi argomenti \vec{w} dato che $f(x)$ gode di queste proprietà.

Dunque ora possiamo usare tecniche analitiche standard per minimizzare questa funzione e una delle tecniche più semplici è quella della discesa lungo il gradiente (appendice A.1) che in questo caso ci fornisce la seguente regola per la variazione dei pesi:

$$w_i = w_i + \Delta w_i = w_i - \varepsilon \frac{\partial E}{\partial w_i}$$

e facilmente troviamo

$$\frac{\partial E}{\partial w_i} = - \sum_{\nu=1}^m [\xi_\nu - f(\vec{x}_\nu \cdot \vec{w})] f'(\vec{x}_\nu \cdot \vec{w}) x_{i\nu}$$

Scrivendola in forma vettoriale abbiamo una regola di variazione dei pesi che discende lungo il gradiente della funzione d'errore, ma che possiamo anche interpretare come una regola d'apprendimento per il Perceptron continuo:

$$\vec{w} = \vec{w} + \varepsilon \sum_{\nu=1}^m [\xi_\nu - f(\vec{x}_\nu \cdot \vec{w})] f'(\vec{x}_\nu \cdot \vec{w}) \vec{x}_\nu \quad (11)$$

Con la definizione dell'errore $\delta_\nu = (\xi_\nu - y_\nu)$ la regola di apprendimento diventa:

$$\vec{w} = \vec{w} + \varepsilon \sum_{\nu=1}^m \delta_\nu f'(\vec{x}_\nu \cdot \vec{w}) \vec{x}_\nu$$

che ne mette in luce la stretta somiglianza con la (8) tenuto conto che quella si intendeva applicata per il singolo esempio. Nel caso particolarmente semplice di una funzione di trasferimento lineare: $y = f(x) = \alpha x + \beta$, è facile vedere che

$$\delta_\nu f'(\vec{x}_\nu \cdot \vec{w}) = \alpha(\xi_\nu - y_\nu)$$

e la somiglianza fra Perceptron continuo e discreto è più marcata: la regola di apprendimento coincide con la (8)¹⁰. La stessa regola si trova anche definendo una diversa funzione d'errore basata sulla teoria dell'informazione, lo vedremo più avanti, al paragrafo 10.2. La principale differenza con il Perceptron discreto è che in quel caso ci sono di solito infinite soluzioni che danno $E(\vec{w}) = 0$ mentre qui si può dimostrare che la soluzione è unica e corrisponde al minimo di $E(\vec{w})$, di norma positivo.

Nel caso generale l'apprendimento per il Perceptron continuo contiene il termine $f'(\vec{x}_\nu \cdot \vec{w})$ che, come si vede in figura 17 per la tangente iperbolica, fa sì che il termine di modifica dei pesi sia significativamente diverso da zero solo se $\vec{x}_\nu \cdot \vec{w} \approx 0$, cioè se l'esempio \vec{x}_ν si trova vicino all'iperpiano separante. Questa caratteristica non era presente nell'algoritmo di apprendimento del

¹⁰il caso di neuroni con funzione di trasferimento lineare, ancorché istruttivo, non è molto interessante dato che si dimostra facilmente che qualsiasi rete di neuroni di questo tipo è sempre equivalente ad un solo neurone.

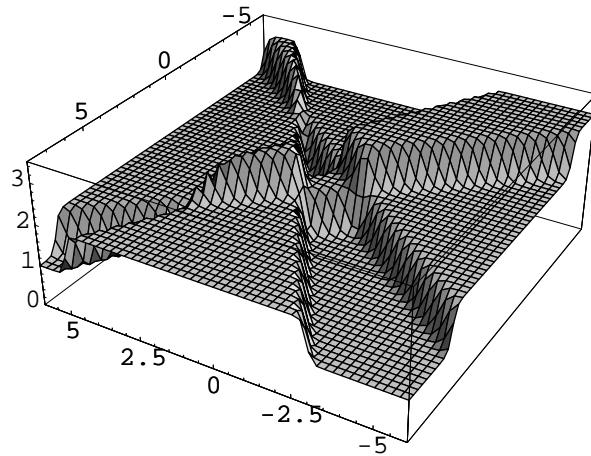


Figura 18: La funzione d'errore $E(w_1, w_2)$ per un problema XOR: si noti che il valore minimo è 1 (e non 0) e che questo valore è raggiunto per diversi valori dei pesi producendo una funzione con diversi minimi relativi.

Perceptron discreto ed è una delle cause che possono rendere l'apprendimento in questo caso incredibilmente lento (rozzamente potremmo dire un ordine di grandezza più lento del Perceptron discreto).

Accenniamo infine al caso in cui la condizione del teorema del Perceptron non sia soddisfatta, ossia quando non esiste un vettore \vec{w} che risolva il problema. Il problema generalizzato è quello di trovare un vettore \vec{w} che renda minimo il numero di esempi classificati male ma esso, al contrario del problema base, si rivela un problema estremamente difficile (in particolare è un problema NP-completo). Si può anche vedere la cosa dal punto di vista dal Perceptron continuo: in questo caso se esiste una soluzione la funzione $E(\vec{w})$ risulta una funzione semplice con un solo minimo che viene facilmente raggiunto con la discesa lungo il gradiente; se invece non esiste un vettore soluzione $E(\vec{w})$ risulta essere una funzione piena di minimi relativi e la discesa lungo il gradiente porta sempre solo al minimo più vicino, si veda per esempio la funzione $E(\vec{w})$ per un Perceptron a due ingressi con il problema dell'XOR in figura 18. Si capisce come la ricerca del minimo assoluto di questa funzione, nascosto in una selva di minimi relativi, diventi un problema estremamente difficile.

La limitazione chiaramente viene dal fatto che il Perceptron implementa una separazione lineare nello spazio di ingresso, tanto per fare un esempio, un neurone con dipendenza quadratica dagli input potrebbe risolvere facilmente il problema dello XOR (sapreste dimostrarlo ?) ma non ci addentriamo in questo argomento.

6 Le reti “feed-forward”

La principale limitazione del Perceptron è dovuta al fatto che per la maggior parte dei problemi (per esempio l’XOR) *non esiste* un insieme di pesi \vec{w} che risolva il problema; ne deriva che nemmeno il teorema del Perceptron vi si applica. Possiamo vedere questa limitazione da un altro punto di vista pensando alla funzione implementata dal Perceptron

$$y = f(\vec{x} \cdot \vec{w})$$

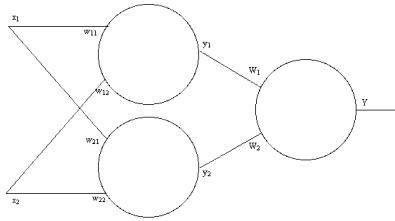


Figura 19: Una semplicissima rete feed-forward con 2 ingressi x_1, x_2 , due neuroni hidden ed un output Y .

come a una funzione $y = y(\vec{x})$ definita nello spazio degli input, cioè nell’iper cubo n -dimensionale H_n . Vediamo subito che le funzioni implementabili dal Perceptron sono un insieme estremamente ridotto delle possibili funzioni continue definite su H_n : in sostanza sono quelle che dividono H_n in due parti con un iper piano di equazione $\vec{x} \cdot \vec{w} = 0$ e nei due semispazi così

ottenuti valgono rispettivamente 1 e -1 più una zona di “transizione” nelle vicinanze dell’iper piano separante. Dunque possiamo vedere le limitazioni del Perceptron come dovute alla semplicità della funzione che riesce ad implementare in H_n . Già nel caso dell’XOR bidimensionale una funzione che riuscisse ad avere una zona di confine con forma quadratica anziché lineare potrebbe risolvere il problema.

Passando dal singolo Perceptron a una rete di Perceptron collegati in modo che l’informazione possa fluire soltanto in una direzione (da cui il nome di reti “feed-forward”, vedi figura 13) scopriamo che vi si può applicare il seguente

Teorema 2 *sia data una funzione $f : \mathbb{R} \rightarrow \mathbb{R}$ continua, monotona, limitata e non costante e sia $C(H_n)$ l’insieme delle funzioni continue $F : H_n \rightarrow \mathbb{R}$ definite sull’iper cubo $H_n = \{\vec{x} \in \mathbb{R}^n : -1 \leq x_i \leq 1, i = 1, \dots, n\}$. Per ogni $F(\vec{x}) \in C(H_n)$ ed $\varepsilon > 0$ esistono delle costanti h, \vec{W}, \vec{w}_i tali che*

$$\left| F(\vec{x}) - \sum_{i=1}^h W_i f(\vec{x} \cdot \vec{w}_i) \right| < \varepsilon \quad \forall \vec{x} \in H_n .$$

Questo teorema, che si può vedere come una generalizzazione dello sviluppo in serie di Fourier (accennato nel paragrafo 13.2) esteso ad una base formata da funzioni sigmoidali $f()$, fornisce alle reti feed-forward la proprietà

di *approssimazione universale*, che, unita a quella di poter costruire una [macchina di Turing](#) (capitolo 6), mostra la loro potenza.

Definiamo una semplice rete feed-forward, vedi figura 19, con n ingressi passati in parallelo ad h neuroni nello strato intermedio. Ognuno di questi h neuroni ha pesi \vec{w}_i con $i = 1, \dots, h$ e produce un output $y_i = f(\vec{x} \cdot \vec{w}_i)$ con funzione di trasferimento sigmoidale, per esempio $f(x) = \tanh(x)$. Questi output intermedi sono convogliati ad un ulteriore neurone che con gli h pesi W_i a sua volta produce l'output $Y = f(\vec{y} \cdot \vec{W})$. Se immaginiamo che solo quest'ultimo neurone abbia una funzione di trasferimento lineare $f(x) = x$ allora¹¹ vediamo che l'output complessivo di questa rete vale

$$Y = f(\vec{y} \cdot \vec{W}) = \vec{y} \cdot \vec{W} = \sum_{i=1}^h y_i W_i = \sum_{i=1}^h W_i f(\vec{x} \cdot \vec{w}_i)$$

che è esattamente la funzione approssimante che compare nel teorema 2.

In altre parole il teorema ci dice, che una rete con un solo strato intermedio (o “hidden”) come quella ora descritta può approssimare *qualsiasi* funzione continua $F(\vec{x})$. Questo risultato cancella in un sol colpo tutte le limitazioni proprie del Perceptron. Il rovescio della medaglia è che questo è un teorema di esistenza ma non dice nulla né sui valori delle costanti, in particolare sul valore di h il numero di neuroni nello strato intermedio, né su come fare a trovarle. In altre parole per queste reti non esiste al momento nessun algoritmo di apprendimento che possa garantire di trovare l'approssimazione che, però, sappiamo esistere. Un altro scotto da pagare è che, come vedremo, per le reti feed-forward non può essere calcolata facilmente né la capacità né d_{VC} per cui non saremo in grado di fare affermazioni sulle capacità teoriche di queste reti.

Possiamo però prendere spunto dal Perceptron per generare un algoritmo di apprendimento. Nel paragrafo 5.2 sul Perceptron continuo infatti abbiamo mostrato che un algoritmo di apprendimento può essere ricavato anche dalla discesa lungo il gradiente della funzione d'errore $E(\vec{w})$ (4). Visto che anche nelle reti feed-forward si considerano funzioni di trasferimento continue potremo far ricorso anche qui alla buona vecchia discesa lungo il gradiente che, per queste reti, prende il nome di algoritmo di *back-propagation*.

Per una rete feed-forward come quella di figura 19 al solito \vec{x}_ν sono gli esempi che vengono presentati e che producono rispettivamente gli output $y_{i\nu}$ nel i -esimo neurone dello strato intermedio e Y_ν nel neurone finale; ξ_ν

¹¹in realtà non occorre rinunciare alla normale $f(x)$ sul neurone finale dato che, per esempio, si possono riscalare i pesi W_i in maniera che l'input sia circa 0 dove la tangente iperbolica è, con buona approssimazione, lineare

sono gli output desiderati. La funzione d'errore da minimizzare è¹²

$$E(\vec{w}_1, \dots, \vec{w}_h, \vec{W}) = \frac{1}{2} \sum_{\nu=1}^m (\xi_\nu - Y_\nu)^2$$

dove Y_ν può essere scritto in diversi modi equivalenti

$$Y_\nu = Y(\vec{x}_\nu) = f(\vec{y}_\nu \cdot \vec{W}) = f\left(\sum_{i=1}^h y_{i\nu} W_i\right) = f\left(\sum_{i=1}^h W_i f(\vec{x}_\nu \cdot \vec{w}_i)\right) \quad (12)$$

La discesa lungo il gradiente prevede per i singoli pesi le seguenti regole di apprendimento

$$\begin{cases} W_i &= W_i - \varepsilon \frac{\partial E}{\partial W_i} \\ w_{ij} &= w_{ij} - \varepsilon \frac{\partial E}{\partial w_{ij}} \end{cases}$$

Iniziamo dal calcolo della prima derivata che è più semplice ed è in sostanza quella del Perceptron continuo

$$-\frac{\partial E}{\partial W_i} = \sum_{\nu=1}^m (\xi_\nu - Y_\nu) f'(\vec{y}_\nu \cdot \vec{W}) y_{i\nu}$$

per la seconda usiamo le regole delle derivate a catena

$$-\frac{\partial E}{\partial w_{ij}} = \sum_{\nu=1}^m (\xi_\nu - Y_\nu) \frac{\partial Y_\nu}{\partial w_{ij}} = \sum_{\nu=1}^m (\xi_\nu - Y_\nu) \frac{\partial Y_\nu}{\partial y_{j\nu}} \frac{\partial y_{j\nu}}{\partial w_{ij}}$$

ricaviamo facilmente che

$$\frac{\partial Y_\nu}{\partial y_{j\nu}} = f'(\vec{y}_\nu \cdot \vec{W}) W_j \quad \frac{\partial y_{j\nu}}{\partial w_{ij}} = f'(\vec{x}_\nu \cdot \vec{w}_j) x_{i\nu}$$

per cui alla fine

$$-\frac{\partial E}{\partial w_{ij}} = \sum_{\nu=1}^m (\xi_\nu - Y_\nu) f'(\vec{y}_\nu \cdot \vec{W}) W_j f'(\vec{x}_\nu \cdot \vec{w}_j) x_{i\nu}$$

Con la già vista definizione dell'errore $\delta_\nu = (\xi_\nu - Y_\nu)$ le regole assumono l'aspetto finale

$$\begin{cases} W_i = W_i + \varepsilon \sum_{\nu=1}^m \delta_\nu f'(\vec{y}_\nu \cdot \vec{W}) y_{i\nu} \\ w_{ij} = w_{ij} + \varepsilon \sum_{\nu=1}^m \delta_\nu f'(\vec{y}_\nu \cdot \vec{W}) W_j f'(\vec{x}_\nu \cdot \vec{w}_j) x_{i\nu} \end{cases}$$

e dato che valgono per tutti i pesi possiamo metterle in forma vettoriale

$$\begin{cases} \vec{W} = \vec{W} + \varepsilon \sum_{\nu=1}^m \delta_\nu f'(\vec{y}_\nu \cdot \vec{W}) \vec{y}_\nu \\ \vec{w}_j = \vec{w}_j + \varepsilon \sum_{\nu=1}^m \delta_\nu f'(\vec{y}_\nu \cdot \vec{W}) W_j f'(\vec{x}_\nu \cdot \vec{w}_j) \vec{x}_\nu \quad j = 1 \dots h \end{cases}$$

¹²con la discesa lungo il gradiente possiamo minimizzare la funzione solo rispetto ai parametri continui $\vec{w}_1, \dots, \vec{w}_h$ e \vec{W} e non rispetto a quello discreto h per cui riterremo fissato a priori il numero di neuroni nello strato intermedio.

Queste regole si possono facilmente generalizzare a reti con più strati intermedi e/o con più neuroni nello stato finale. Osserviamo che nelle regole di apprendimento ora trovate per poter calcolare la variazione dei pesi occorre aver presentato alla rete tutti gli m esempi \vec{x}_ν , dato che nei calcoli appare la somma su tutti i ν . Questo modo di procedere, detto modo *batch*, pur analiticamente corretto è raramente usato perché piuttosto lento da eseguire. È molto più frequentemente usato il modo *incrementale* nel quale si sceglie un esempio a caso, lo si presenta alla rete e si aggiornano subito i pesi con le

$$\begin{cases} \vec{W} = \vec{W} + \varepsilon \delta_\nu f'(\vec{y}_\nu \cdot \vec{W}) \vec{y}_\nu \\ \vec{w}_j = \vec{w}_j + \varepsilon \delta_\nu f'(\vec{y}_\nu \cdot \vec{W}) W_j f'(\vec{x}_\nu \cdot \vec{w}_j) \vec{x}_\nu \end{cases} \quad (13)$$

È facile convincersi che i due algoritmi coincidono nel limite $\varepsilon \rightarrow 0$ (dimostratelo per esercizio).

Notiamo che se per la funzione di trasferimento scegliamo $f(x) = \tanh(x)$ il calcolo numerico delle derivate sarà molto semplificato dato che, in questo caso, $f'(x) = 1 - f(x)^2$ e per avere il valore della derivata basterà calcolare un quadrato.

Questa regola di apprendimento porta sempre ad un minimo di E , il problema è che questo minimo può essere relativo perché non è difficile convincersi che la funzione E ha molti minimi relativi anche quando la funzione che si vuole imparare ha un minimo assoluto con $E \approx 0$. Questo si vede già nel semplice esempio dell'XOR nel quale, con una rete di 3 neuroni come in figura 19, si possono trovare parecchie soluzioni diverse separate fra di loro, nello spazio dei pesi, da regioni nelle quali la funzione E assume valori alti (sapreste dimostrarlo?) come rappresentato in figura 18. Le cose peggiorano al crescere delle dimensioni della rete e, di conseguenza del numero dei pesi, infatti più pesi ci sono e più è facile finire in un minimo relativo dato che il loro numero cresce esponenzialmente al crescere del numero di pesi. Questa proprietà viene chiamata in modo colorito: 'maledizione delle dimensioni' (*curse of dimensionality*).

Traiamo spunto dall'esempio dell'XOR per notare che condizione necessaria affinché una rete possa trovare una soluzione è che la rappresentazione fra il primo e secondo strato sia *fedele*. Ciò significa che esempi nello spazio degli input, per i quali l'output desiderato sia diverso, devono essere rappresentati nello spazio intermedio da configurazioni diverse. In altre parole dati due esempi \vec{x}_ν e \vec{x}_μ se $\xi_\nu \neq \xi_\mu$ allora deve essere necessariamente $\vec{y}_\nu \neq \vec{y}_\mu$ altrimenti lo strato successivo non potrà produrre output diversi per questi esempi. In figura 20 ci sono tre esempi per un problema XOR per la rete di 3 neuroni di figura 19. In ogni riga c'è un caso e vi sono rappresentati lo spazio di input ($x_1 x_2$) e quello al livello intermedio ($y_1 y_2$). Nello spazio $x_1 x_2$, oltre ai quattro input possibili, sono rappresentate le separazioni dello spazio di input fatte dai neuroni dello strato intermedio. Nel primo caso si

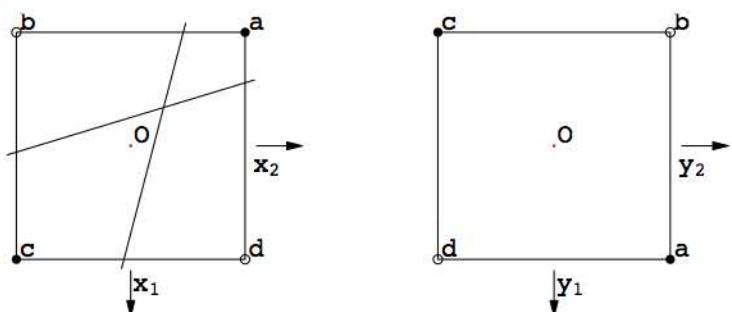
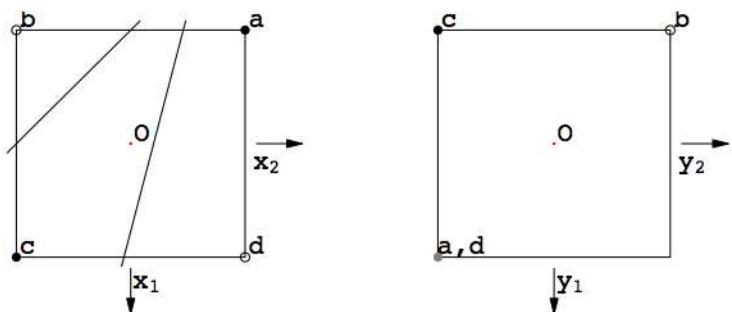
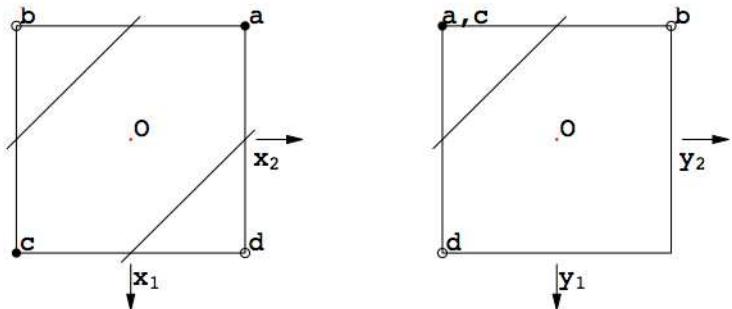


Figura 20: Rappresentazioni nell'XOR a due ingressi di figura 19: le tre copie di figure corrispondono ad una rappresentazione fedele (prima riga), ad una non fedele, ed a una fedele ma che non permette di trovare la soluzione. Il primo quadrato di ogni esempio rappresenta la spazio di input x_1x_2 mentre il secondo rappresenta lo spazio dei neuroni intermedi, ovverosia l'input per il neurone finale; maggiori dettagli nel testo.

vede che i quattro possibili esempi, marcati per comodità a, b, c , e d , sono rappresentati nello spazio intermedio in 3 sole configurazioni, quella $-1, -1$ non si presenta mai. In questo caso il neurone finale può separare i patterns allo stadio intermedio risolvendo il problema. Nelle righe successive sono rappresentati altri due casi che si possono verificare che non permettono di trovare soluzione al problema. Nell'esempio di mezzo la variazione della classificazione di uno dei due neuroni in ingresso fa sì che il pattern a non viene più classificato con il pattern c ma bensì con quello d e dunque al livello intermedio gli esempi a e d vengono classificati insieme pur avendo un output desiderato diverso. Questo fa sì che non esistano funzioni dell'hidden layer nell'output che possano classificare correttamente entrambi gli esempi a e d . Nella terza riga c'è stata un ulteriore variazione nelle funzioni dell'input layer ed ora gli esempi a e d non vengono più classificati insieme, nondimeno si vede subito che non esiste funzione per l'output layer che possa separare correttamente i pattern dato che abbiamo riprodotto, al livello intermedio, il problema di partenza: l'XOR.

In conclusione al secondo strato necessariamente occorre avere una rappresentazione fedele ma questa condizione non è sufficiente come mostra l'ultimo esempio. Inoltre cercare una soluzione minimizzando E non tiene minimamente conto di queste condizioni né ci sono metodi semplici per imporle alla rete.

Infine menzioniamo un'altra difficoltà nell'uso di queste reti: in generale, dato un problema, non è noto quanti neuroni al minimo occorrono nello strato intermedio per arrivare alla soluzione. Può accadere che si inizi ad applicare la back-propagation ad una rete e che, con quel numero di neuroni nello strato intermedio, il problema non abbia soluzione (per esempio: nel caso dell'XOR una rete con 2 input e 1 + 1 neuroni in cascata non può trovare la soluzione). Magari sarebbe bastato aggiungere un solo neurone nello strato intermedio per poter arrivare alla soluzione.

Tutte queste considerazioni mostrano come l'apprendimento con la back-propagation soffra di gravi carenze strutturali, nondimeno è uno degli algoritmi più usati per la sua facilità d'uso, specialmente su reti con pochi neuroni.

6.1 La “back-propagation” in pratica

Diamo qualche consiglio generale per programmare e testare l'algoritmo di back-propagation in qualche caso semplice.

Per prima cosa definiremo un problema che si vuol far “apprendere” alla rete. Potrebbe andare bene una funzione Booleana di 5–10 ingressi. Alternativamente potremmo considerare delle serie temporali e cioè data una serie di n bit prevedere l' $n + 1$ -esimo. Poi si ripete il procedimento: anche qui si tratta in pratica di una funzione Booleana che però è definita in modo diverso. Infine, per i più ambiziosi, in rete si trovano facilmente

decine di problemi ben studiati e analizzati (per esempio riconoscimento di caratteri scritti a mano, formazione di diagnosi da una serie di sintomi etc.). Di solito questi problemi hanno molti input per cui l'apprendimento rischia di essere molto lungo per cui il mio consiglio è di mettere a punto un programma su un problema semplice e poi eventualmente provare a usarlo su problemi più difficili.

Deciso un problema occorre definire un insieme di esempi e di output desiderati: in sostanza si tratta di una matrice $n \times m$.

Una rete con n neuroni nello strato di ingresso, h in quello hidden e 1 in quello di output completamente connessi fra di loro è definita univocamente dal valore dei relativi pesi. Dunque mi occorre un array $n \times h$ per il primo strato (ricordando che ci sono anche le soglie) e di un vettore h per il secondo strato.

Schematizziamo così un programma che esegue l'algoritmo:

1. inizializza gli esempi e relativi output desiderati e i pesi a valori casuali (tipicamente piccoli numeri diversi da 0);
2. si genera un numero casuale $1 \leq \nu \leq m$ per scegliere l'esempio \vec{x}_ν , da dare alla rete;
3. si propaga l'input in avanti e si calcola l'output Y_ν (fase “forward”);
4. si calcola δ_ν e lo si propaga all'indietro modificando i pesi (fase “backward”, si ricordi che calcolare le derivate è facile se si usa la funzione $\tanh(x)$);
5. con il nuovo valore dei pesi si ricalcola E e lo si stampa (questo calcolo non è necessario all'apprendimento vero e proprio, serve solo per far vedere come procede l'algoritmo);
6. si verifica un criterio di stop (vedi sotto) e, se non raggiunto, si ritorna al punto 2.

Non c'è un criterio di stop che vada bene per tutti i casi; deve essere definito da un certo numero di regole che, per esempio, potrebbero essere:

- ci si ferma se $E < cost$. dove la costante potrebbe essere 0.2: in questo caso si è trovata soluzione al problema;
- ci si ferma se il numero di iterazioni è troppo grande, in questo caso non si è trovata la soluzione;
- ci si ferma se la variazione di E dal passo successivo è troppo piccola (per esempio 10^{-8}): questo significa che non si stanno facendo miglioramenti e/o che siamo in una zona “piatta”;

- ci si ferma se il modulo di $\Delta\vec{w}$ è troppo piccolo (per esempio 10^{-8}): questo significa che non ci si sta muovendo nello spazio dei pesi.

Questi sono criteri base che possono andare bene per i primi semplici test, in letteratura si trovano moltissimi altri esempi, anche molto raffinati.

Un caso tipico che si incontra è quello di una funzione E fatta a “grandaia” in questi casi $\Delta\vec{w}$ è grande ma lo spostamento reale nello spazio dei pesi può essere ridottissimo. Questo problema si può curare in vari modi: per esempio aggiungendo un termine di “momento” alla regola di apprendimento, chi è interessato può andare a cercare in qualcuno dei libri suggeriti.

Valori tipici delle costanti per iniziare a fare dei test possono essere $n = h = 5$, $\beta = 1$ ed $\varepsilon = 10^{-4}$.

Infine alcuni consigli ancora più pratici:

- cominciate da un problema veramente semplice per mettere a punto la back-propagation. Per esempio riconoscere se un numero scritto in forma binaria è pari o è una potenza di due, riconoscere se almeno metà degli input sono a 1 etc. etc. Questi problemi sono banali e dato che è facile trovare una soluzione esatta esaminando la rete posso verificare se l'algoritmo funziona.
- per un programma banale come quelli sopra provare a dare come punto di partenza la soluzione nota e verificare che non si muova da lì.
- scrivere una funzione a parte per il calcolo di E da chiamare ad ogni modifica dei pesi nella fase di messa a punto del programma per vedere se in effetti E diminuisce durante l'apprendimento.
- per problemi non proprio banali, a parità di numero di pesi, una rete con più strati si dimostra di solito più efficiente di una rete con solo due strati (le regole di apprendimento (13) si estendono facilmente).
- etc.etc.

6.2 Caratteristiche delle reti “feed-forward”

In breve alcune caratteristiche delle reti feed-forward e del relativo algoritmo di apprendimento:

- sono flessibili: con l'apprendimento si adattano facilmente a nuove condizioni,
- sono efficaci ma soprattutto per problemi “piccoli”, crescendo n si incontra la già menzionata “curse of dimensionality” che significa che mentre reti con pochi neuroni trovano facilmente una soluzione accettabile reti grandi affogano facilmente nel mare dei minimi relativi di E ;

- l’architettura di queste reti deve essere decisa a priori ed è “congelata” (per esempio può darsi che l’ h scelto sia troppo piccolo per permettere di trovare una soluzione),
- l’algoritmo di apprendimento è sempre della forma $\vec{w} = \vec{w} + \varepsilon \delta_\nu \vec{x}_\nu$; ne segue che in sostanza esso è locale e ricorda da vicino la regola di Hebb estesa,
- nondimeno è un algoritmo fin troppo semplice che soffre il problema dei minimi locali.

Concludiamo con una serie di domande su queste reti, la maggior parte delle quali sono a tutt’oggi senza risposta:

- data una rete di una certa dimensione quali funzioni è in grado di approssimare e quali no (in un certo senso la “parte mancante” del teorema di approssimazione universale);
- fra le funzioni che possono essere implementate quali possono essere “apprese” da esempi e quanti esempi occorrono per apprenderle correttamente;
- quanti neuroni al minimo servono per implementare una data funzione;
- qual è un algoritmo per il learning e quanto bene funziona (che probabilità ha di imparare, quanti esempi servono etc.);
- cosa succede se i pesi sono limitati, per esempio in precisione (si noti che con pesi limitati anche il teorema del Perceptron non è più valido);
- quali condizioni devono essere rispettate per avere una buona generalizzazione.

Almeno a quest’ultima domanda sono state date delle parziali risposte, quasi sorprendentemente perché sembrava una delle domande più difficili. La studieremo nei prossimi paragrafi.

7 La capacità del Perceptron

Presentiamo ora alcune idee e risultati della recente formulazione matematico-statistica dell'apprendimento (COLT - COmputational Learning Theory)¹³ ma piuttosto che introdurre il problema formalmente ci avvarremo del Perceptron che, essendo semplice e fornito di una chiara interpretazione geometrica, ci permette di addentrarci in questo campo seguendo ragionamenti semplici. Dunque useremo il Perceptron per definire idee e concetti che poi potremo esportare su reti più generali.

Piuttosto che investigare i limiti del Perceptron studiando i problemi che non riesce a risolvere, per esempio l'XOR, un approccio più fruttuoso è quello di definire una misura quantitativa delle capacità del Perceptron che poi potremo applicare anche ad altre reti.

Iniziamo osservando che dati $m \leq 2^n$ esempi posso definire al più 2^m diversi problemi (in pratica ogni problema corrisponde a una funzione binaria di m input). In generale un Perceptron non riuscirà a implementarli tutti ma solo un numero $C(m, n)$ che chiaramente dipende dalle dimensioni dello spazio di input n (compresa la soglia) e dal numero di esempi m ; ovviamente $C(m, n) \leq 2^m$.

Questo lo si capisce facilmente ripensando agli esempi fatti per il Perceptron a 2 ingressi e riprodotti in figura 16: in questo caso c'erano $m = 4$ esempi che potevano dare origine a 2^4 problemi dei quali solo 14 erano risolvibili dal Perceptron. Dunque in questo caso diremo che il numero di funzioni diverse implementabili dal Perceptron con 3 pesi e 4 esempi è $C(4, 3) = 14$. Se invece ci limitiamo a $m = 3$ esempi (assumendo che, per esempio, l'input $(1, 1)$ non si presenti mai) facilmente si vede che il Perceptron può implementare tutti i $2^3 = 8$ possibili problemi, cioè $C(3, 3) = 8$.

Studieremo la quantità $C(m, n)$ e più in particolare il rapporto

$$\frac{C(m, n)}{2^m} \leq 1$$

che è la frazione di problemi risolvibili da un Perceptron. Prima di procedere alcune osservazioni:

- La quantità $C(m, n)$ che ci accingiamo a calcolare non è unicamente legata al Perceptron. $C(m, n)$ è in realtà una quantità geometrica più generale, che da il numero di modi diversi di separare m punti in uno spazio \mathbb{R}^n con un iperpiano che passa per l'origine.
- Abbiamo visto che per $m = 3$ punti nel piano il Perceptron può implementare tutti i possibili 8 problemi ma si verifica subito che questo

¹³I primi lavori del matematico Vladimir Vapnik, vedi figura 23, sono del 1971, poi proseguono con Leslie Valiant dal 1984, vedi su: http://en.wikipedia.org/wiki/Computational_learning_theory

non è vero nel caso particolare nel quale i 3 esempi siano allineati (per esempio —o—?—o—). A questo e simili casi particolari si può ovviare con la richiesta aggiuntiva che i punti che considereremo siano in *posizione generale*. Questo vuol dire che non ci siano allineamenti casuali fra i vari punti¹⁴.

- Occorre specificare esattamente cosa si intende per funzioni diverse implementate dal Perceptron: di solito esistono molti valori di pesi che danno origine alla stessa dicotomia dei punti. Noi diremo che due pesi \vec{w}_1 e \vec{w}_2 sono diversi solamente se implementano dicotomie diverse, in altre parole se esiste almeno un ν tale che $\text{sgn}(\vec{x}_\nu \cdot \vec{w}_1) \neq \text{sgn}(\vec{x}_\nu \cdot \vec{w}_2)$.

Calcoliamo ora $C(m, n)$ usando una relazione di ricorrenza; immaginiamo di avere già disposto $m - 1$ punti e di conoscere tutti i $C(m - 1, n)$ modi diversi di mettere un iperpiano tra di essi. Ora aggiungiamo un m -esimo punto e calcoliamo $C(m, n)$. Osserviamo che gli iperpiani $C(m - 1, n)$ si dividono in due categorie: quelli che possono essere “aggiustati” e fatti passare per l’ m -esimo punto e quelli che invece non possono passare per il nuovo punto senza dare origine ad una diversa separazione fra gli $m - 1$ punti precedenti. Chiamiamo il numero di questi iperpiani rispettivamente r_1 e r_2 ; ovviamente $r_1 + r_2 = C(m - 1, n)$. Volendo calcolare ora $C(m, n)$ osserviamo che gli r_2 iperpiani che non possono esser fatti passare per l’ m -esimo punto danno lo stesso numero di separazioni dopo l’aggiunta del nuovo punto. Invece gli r_1 iperpiani che possono passare per il nuovo punto daranno origine ciascuno a 2 dicotomie a seconda che si lasci l’ m -esimo punto da una parte o dall’altra dell’iperpiano. Dunque abbiamo dimostrato che

$$C(m, n) = 2r_1 + r_2 = C(m - 1, n) + r_1 .$$

Per calcolare r_1 basta osservare che questo numero è quello di iperpiani separanti $m - 1$ punti con la richiesta aggiuntiva che l’iperpiano passi per un punto esterno dato. Ma questo vincolo è di fatto equivalente a muoversi in uno spazio con una dimensione in meno per cui $r_1 = C(m - 1, n - 1)$ col che abbiamo trovato la relazione di ricorrenza voluta

$$C(m, n) = C(m - 1, n) + C(m - 1, n - 1) .$$

Applicando ripetutamente questa relazione troviamo

$$\begin{aligned} C(m, n) &= \\ C(m - 1, n) + C(m - 1, n - 1) &= \\ C(m - 2, n) + 2C(m - 2, n - 1) + C(m - 2, n - 2) &= \\ C(m - 3, n) + 3C(m - 3, n - 1) + 3C(m - 3, n - 2) + C(m - 3, n - 3) &= \dots \end{aligned}$$

¹⁴Più esattamente si può definire un insieme di punti in posizione generale in uno spazio a n dimensioni quando tutti i sottinsiemi formati da n o meno punti siano linearmente indipendenti. Noi considereremo solo insiemi di punti in posizione generale che sono più semplici da trattare. Una parziale giustificazione deriva dall’osservazione che, se si scelgono punti a caso in \mathbb{R}^n , questa condizione è di gran lunga la più probabile.

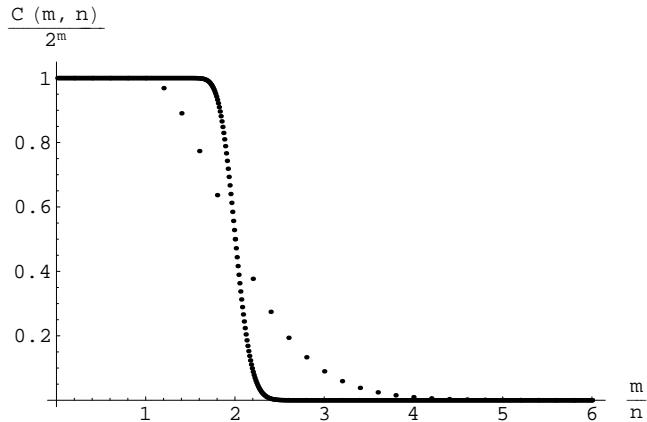


Figura 21: $\frac{C(m,n)}{2^m}$ calcolato per $n = 5$ (punti radi) e $n = 100$ (punti densi). Per poter sovrapporre agevolmente le curve in ascissa c'è il rapporto m/n .

dalle quali osserviamo che dopo 3 applicazioni m si è ridotto a $m-3$ in tutti i termini mentre n varia da n a $n-3$ (nella struttura dei coefficienti si riconosce facilmente il triangolo di Pascal). È chiaro che con $m-1$ applicazioni della regola di ricorrenza arriviamo alla

$$\begin{aligned} C(m, n) &= \binom{m-1}{0} C(1, n) + \binom{m-1}{1} C(1, n-1) + \dots \\ &\dots + \binom{m-1}{k} C(1, n-k) + \dots \end{aligned}$$

dove il coefficiente binomiale indica che a $C(1, n-k)$ si è arrivati in $\binom{m-1}{k}$ modi diversi applicando k volte il secondo termine della relazione di ricorrenza, quello che diminuisce n . Quale sarà l'ultimo termine? Chiaramente $C(m, n)$ è definito solo in uno spazio con $n \geq 1$ dimensioni per cui nella formula trovata dovremo avere $n-k \geq 1$ e cioè $k \leq n-1$. Osserviamo ora che $C(1, n) = 2$ per tutti gli $n \geq 1$ per cui possiamo concludere¹⁵

$$C(m, n) = 2 \sum_{k=0}^{n-1} \binom{m-1}{k} \quad (14)$$

e la frazione di problemi risolubili dal Perceptron si può scrivere

$$\frac{C(m, n)}{2^m} = \sum_{k=0}^{n-1} \binom{m-1}{k} \frac{1}{2^k} \frac{1}{2^{m-1-k}} \quad (15)$$

¹⁵Osserviamo che se $m < n$ la somma dovrebbe fermarsi a $m-1$ ma di questo si tiene conto implicitamente nella relazione trovata assumendo che $\binom{m-1}{k} = 0$ quando $k > m-1$.

che è una somma di probabilità binomiali con $p = 1 - p = 1/2$ (la distribuzione binomiale è ripetuta brevemente nell'appendice A.2) e la figura 21 ne mostra la dipendenza da m quando $n = 5$ e 100 . Osserviamo che al crescere di n la curva si fa più ripida. Un'altra caratteristica che si desume dalla (15) è che per $m \leq n$ la somma si estende su tutte le probabilità binomiali e dunque vale 1 e cioè $C(m, n) = 2^m$. Questo ci dice che fintantoché il numero di esempi $m \leq n$ il Perceptron è in grado di implementare tutte le funzioni Booleane possibili sugli m punti e questa è una proprietà puramente geometrica dello spazio degli input.

7.1 La capacità del Perceptron

Il rapporto (15) è una somma, più o meno estesa, degli elementi di una distribuzione binomiale e studiare come varia $\frac{C(m,n)}{2^m}$ al variare di m significa variare la distribuzione binomiale mantenendo fisso l'ultimo elemento della sommatoria (15): ovviamente per $m \leq n$ la distribuzione binomiale è sommata per intero e al crescere di m , ossia della sua coda destra, diminuisce la frazione di distribuzione che entra nella sommatoria. È facile convincersi che se sommiamo esattamente metà della distribuzione la sua somma varrà metà del totale cioè $\frac{1}{2}$ dato che la binomiale con $p = 1 - p = 1/2$ è simmetrica rispetto al suo valore centrale. La condizione per cui questo si verifica è che vi sia lo stesso numero di addendi nella prima e nella seconda parte della sommatoria. La sommatoria dell'intera binomiale ha $m - 1 + 1 = m$ termini, nella parte che viene sommata nella (15) ve ne sono n , la condizione perché ve ne siano altrettanti nella seconda è che valga $m - n = n$ da cui si ricava $m = 2n$ con la quale

$$\frac{C(2n, n)}{2^m} = \frac{1}{2} .$$

Nella figura 21 il valore $\frac{1}{2}$ viene raggiunto per $m = 2n$, questo valore viene chiamato la *capacità* del Perceptron.

Un altro modo di vedere questa definizione si ha interpretando la quantità $\frac{C(m,n)}{2^m}$ come la probabilità che un problema con m esempi scelto a caso sia risolubile da un Perceptron. In questo caso la capacità è quel valore di m che rende $1/2$ questa probabilità.

Osserviamo che entrambe le quantità definite: la capacità ($m = 2n$) e il valore fino al quale il Perceptron può implementare qualsiasi funzione ($m \leq n$)¹⁶ in sostanza definiscono i valori di m ai quali il numero di funzioni implementabili dal Perceptron si riduce rispetto al limite superiore 2^m .

¹⁶fra poco – nel paragrafo 7.3 – sarà la dimensione di Vapnik e Cervonenkis d_{VC}

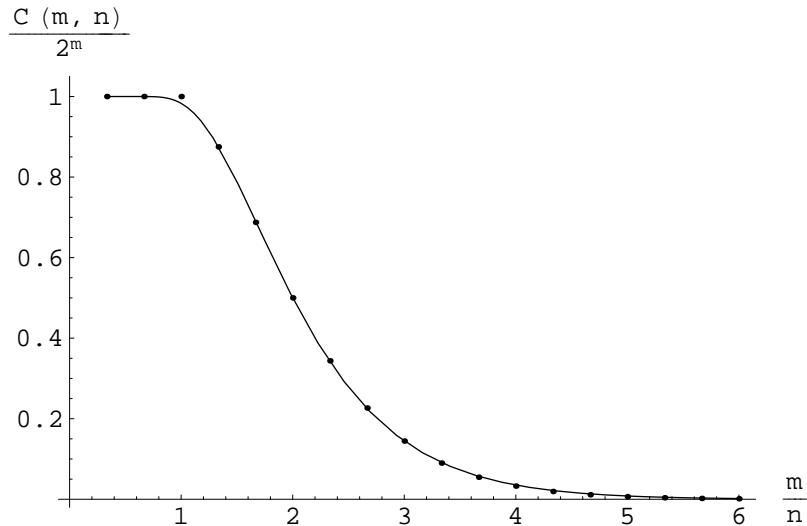


Figura 22: Approssimazione per la (14) calcolata per $n = 3$.

7.2 Approssimazione Gaussiana di $C(m, n)$

Possiamo approssimare la (15) per grandi m approssimando la binomiale con una Gaussiana come fatto, per sommi capi, nell'appendice A.2. La nostra distribuzione binomiale ha parametri $N = m - 1$ e $p = 1/2$, che, per la Gaussiana approssimante $G_{\mu\sigma}(k)$, danno rispettivamente $\mu = \frac{m-1}{2}$ e $\sigma = \sqrt{\frac{m-1}{2}}$. A questo punto possiamo approssimare la (15) con $\int_{-\infty}^{n-\frac{1}{2}} G_{\mu\sigma}(k) dk$, e con la (53), sostituendo i nostri valori troviamo

$$\lim_{m,n \rightarrow \infty} \frac{C(m, n)}{2^m} = \frac{1}{2} \left[1 + \operatorname{erf} \left(\frac{2n - m}{\sqrt{m-1}\sqrt{2}} \right) \right]$$

e dalla figura 22 si vede che già per $n = 3$ e m compreso fra 1 e 18 l'approssimazione è ottima.

7.3 La dimensione di Vapnik e Cervonenkis d_{VC}

Il massimo valore del numero di esempi m per il quale la funzione implementata da una certa rete, in questo caso il semplice Perceptron, vale 2^m è un valore importante per quello che segue, si generalizza a qualsiasi tipo di rete feed-forward, e viene chiamata *dimensione di Vapnik e Cervonenkis* d_{VC} dal nome dei due matematici che nel 1971, vedi figura 23, hanno iniziato lo studio matematico-statistico dell'apprendimento cui



Figura 23: Vladimir N. Vapnik

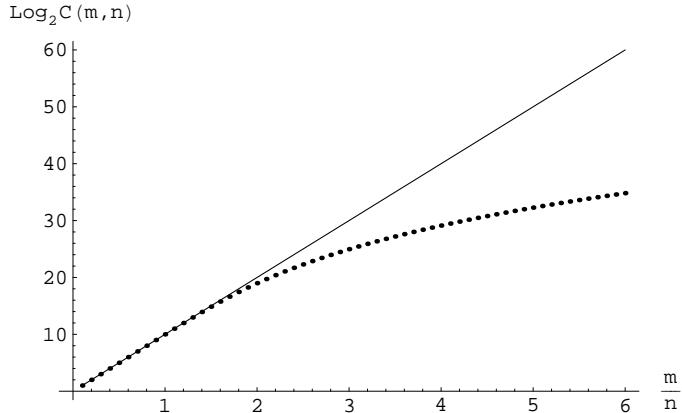


Figura 24: Growth function per il Perceptron calcolata per $n = 10$.

abbiamo accennato. Ora possiamo dire che per il Perceptron $d_{VC} = n$. Sempre per i risultati di questi autori il numero di funzioni che una certa rete può implementare in funzione di m è detta “growth function” $G(m)$ e per il Perceptron essa vale $C(m, n)$, con questa definizione si ha che d_{VC} è il massimo valore di m per il quale $G(m) = 2^m$. La figura 24 mostra l’andamento di $G(m)$ per un Perceptron a 10 ingressi confrontata con il limite superiore 2^m . Notiamo che la funzione $G(m)$ è limitata dall’architettura della rete; per esempio se consideriamo una *qualsiasi* rete di neuroni con W pesi diversi ognuno dei quali può assumere solo k valori diversi allora ci saranno al più k^W funzioni implementabili e

$$G(m) \leq k^W \quad (16)$$

e si può dimostrare anche che vale il limite superiore

$$G(m) \leq m^{d_{VC}} + 1 . \quad (17)$$

8 Il problema della generalizzazione

Nel caso di una rete con 2 ingressi gli input possibili erano solamente 4 ed in ogni problema di apprendimento era facile definire l’output desiderato per tutti i 4 input possibili. Ma già nel caso di un’immagine digitale binaria minuscola di 10×10 pixel gli input possibili sono $2^{100} \simeq 1.3 \cdot 10^{30}$ ed è impensabile di specificare l’output desiderato per tutti gli input. Dunque in tutti i casi realistici avverrà che $m \ll 2^n$ e cioè che il numero degli esempi sia un sottinsieme ridottissimo di tutti gli input possibili¹⁷. Questo ci pone dinanzi un problema che finora era passato inosservato: quando vogliamo insegnare

¹⁷In termini statistici possiamo pensare ai 2^n input come alla *popolazione* e agli m esempi come a un suo *campione*

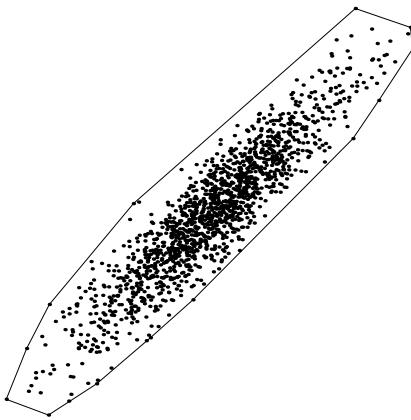


Figura 25: Esempio di un insieme di punti nel piano con i punti che fanno da “picchetti” (in realtà è il “convex hull” dei punti).

a una rete a riconoscere le ‘a’ possiamo solo dargli un piccolo sottinsieme di tutte le possibili a e su queste costruiamo una funzione d’errore che viene minimizzata nell’apprendimento. Ma come risponderà la rete su una a che non faceva parte dell’insieme \mathcal{E} degli esempi ? questo è il problema della *generalizzazione*. Questa ed altre quantità relative all’apprendimento da esempi si studiano nella Computational Learning Theory della quale accenneremo a qualche concetto e daremo un risultato sulla generalizzazione.

A prima vista il problema di trovare una rete che generalizza bene può sembrare impossibile: se con l’apprendimento si è arrivati ad azzerare la funzione d’errore E significa che gli m esempi che sono stati appresi perfettamente. Una rete con n ingressi realizza sempre una delle possibili 2^{2^n} funzioni binarie di n ingressi ma di queste v’è n’è ben 2^{2^n-m} che danno la risposta voluta sui nostri m esempi. Dunque se ammettiamo che la funzione che cerchiamo (quella che riconosce perfettamente tutte le a) sia unica è chiaro che la probabilità di aver trovato proprio quella fra le 2^{2^n-m} compatibili con i nostri esempi e’ praticamente nulla e da questo punto di vista una corretta generalizzazione appare un problema senza speranza.

La spiegazione è che non tutte le funzioni sono ugualmente probabili, alcune sono molto “difficili” (in realtà hanno alta complessità algoritmica) e, per essere apprese correttamente hanno bisogno di $O(2^n)$ esempi. Altre sono più “facili” e bastano pochi esempi per essere apprese: per esempio la funzione che dice se un numero scritto in forma binaria è pari o dispari può essere appresa perfettamente da pochissimi esempi.

Una spiegazione intuitiva di come possa accadere che in certi casi si possa apprendere correttamente una funzione da pochi esempi si può avere con il seguente esempio. Supponiamo che tutti gli esempi che desidero apprendere, quelli cioè per i quali voglio che l’output sia 1, siano un sottinsieme dei 2^n

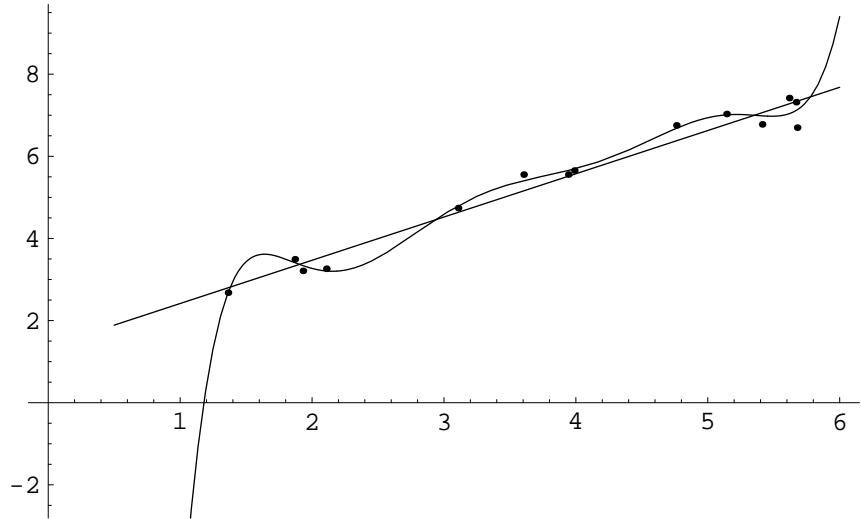


Figura 26: Esempio di overfitting: dati fittati con una retta e con un polinomio di grado 7.

input possibili, sia pur numeroso, ma molto circoscritto in \mathbb{R}^n . Li possiamo immaginare come una nuvola compatta di punti tutti vicini e adiacenti. In altre parole se prendiamo un punto vicino al centro di questo insieme e da questo ci muoviamo in varie direzioni tutti i punti che incontriamo siano punti dell'insieme. Se le cose stanno così, immaginiamo adesso di prendere come m esempi di \mathcal{E} dei punti che stiano al confine di questa nuvola e che la circondino completamente in tutte le direzioni; possiamo immaginare i punti di \mathcal{E} come i picchetti di confine della nostra nuvola, vedi figura 25. In questo caso apprendendo il nostro insieme di esempi avremo definito in pratica il confine di separazione e, una volta terminato l'apprendimento, non solo i punti di confine ma tutti i punti all'interno della nuvola saranno classificati correttamente, anche se abbiamo fatto l'apprendimento da un sottinsieme minuscolo dei punti della nuvola.

Questo esempio non deve trarre in inganno infatti in realtà è assai difficile sapere se gli esempi formano un insieme compatto in \mathbb{R}^n , e, anche nel caso, è tutt'altro che banale scegliere gli esempi significativi (i picchetti) ma lo scopo dell'esempio era solo quello di mostrare un caso nel quale la generalizzazione è teoricamente possibile e comprensibile.

Per illustrare un altro aspetto della generalizzazione torniamo al vecchio esempio del fit di una retta: se abbiamo dei dati che vengono da una relazione lineare e sono affetti da errore è chiaro che fittandoli con una retta troveremo dei coefficienti per la retta che non azzereranno il χ^2 . Se però decidiamo di fittare i nostri dati con un polinomio di grado elevato facilmente troveremo dei parametri che diano $\chi^2 \approx 0$ ma non è detto che le cose vadano meglio.

Volendo prevedere un nuovo valore della y data la x è ragionevole aspettarsi che il fit con la retta darà un valore molto più vicino al vero di quello del polinomio, vedi figura 26. Per il polinomio diremo che c'è stato overfitting dei dati. Questo esempio mostra da un altro punto di vista il concetto di generalizzazione: l'estrapolazione a valori nuovi del nostro fit dei dati e per molti versi ricorda quello che succede in una rete neurale. In sostanza il messaggio è che aggiungere parametri (neuroni e pesi) può rendere più facile l'apprendimento ma non garantisce una buona capacità di previsione della rete, anzi tende a peggiorarne le prestazioni. D'altro canto è chiaro che se voglio fittare dati che seguono una legge quadratica con una retta posso farlo ma la mia generalizzazione non sarà mai buona, per cui è importante riuscire a trovare il numero minimo di parametri per generalizzare meglio i miei dati.

Questo semplice esempio ci mostra anche che deve esistere una relazione fra numero di esempi minimo e funzione da apprendere: chiaramente se voglio costruire una retta mi servono almeno 2 esempi, 3 se devo costruire una relazione quadratica etc. In generale più complessa è la funzione da apprendere dagli esempi maggiore dovrà essere il loro numero.

Ritornando alle reti neurali si ha overfitting quando una rete riconosce tutti e soli gli esempi di \mathcal{E} senza alcuna generalizzazione. Una tale rete è molto facile da costruire: basta mettere nello strato intermedio m neuroni, cioè uno per ogni esempio \vec{x}_ν , ed un neurone di uscita. I pesi del ν -esimo neurone dello strato intermedio hanno i primi $n - 1$ pesi uguali alle coordinate dell'esempio \vec{x}_ν e soglia $-n+1$. Data che il prodotto scalare fra due vettori con n componenti ± 1 può assumere solo uno degli n valori $-n, -n+2, \dots, n-2, n$ si vede che quando si presenta all'input l'esempio \vec{x}_μ tutti i neuroni dello strato intermedio avranno output -1 meno quello μ che darà $+1$ (il suo sarà l'unico prodotto scalare a valere $+n$). Lo strato successivo agisce come un OR logico a m ingressi, ha pesi $\vec{W} = (1, 1, \dots, 1, m-1)$, e da 1 se almeno uno dei suoi input vale 1 e dunque questa rete riconosce tutti e soli gli esempi dati. Questo tipo di rete viene detto della *grandmother cell representation* perché ogni neurone è incaricato di memorizzare un unico esempio (come se ci fosse un neurone del nostro cervello dedicato al solo riconoscimento della nonna) e dà apprendimento perfetto ma generalizzazione

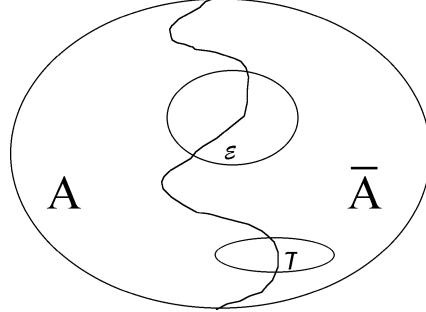


Figura 27: Apprendimento delle A: lo spazio dei 2^n input è ripartito in due sottinsiemi: quello delle A e quello delle non-A, ci sono poi l'insieme degli esempi \mathcal{E} e quello di test \mathcal{T} .

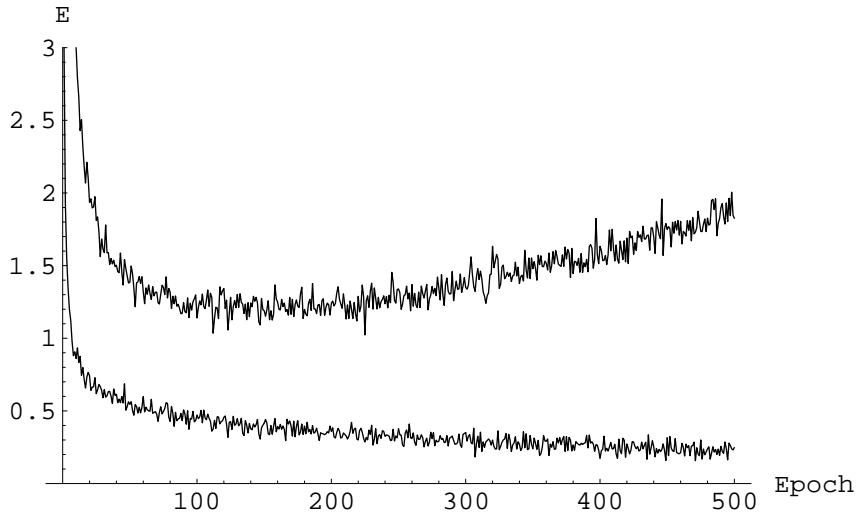


Figura 28: Errore sugli esempi \mathcal{E} (curva inferiore) e sull’insieme di test \mathcal{T} . Il momento giusto per fermare l’apprendimento è quando l’errore su \mathcal{T} raggiunge il minimo.

nulla.

Un fattore determinante nella generalizzazione è il numero totale di pesi della rete: più sono i pesi maggiore sarà l’adattabilità della rete e di conseguenza maggiore la probabilità di avere overfitting. In un mondo ideale bisognerebbe sapere quanti esempi sono necessari per descrivere sufficientemente la funzione che cerchiamo e successivamente costruire una rete con un numero di pesi adatto a descrivere compiutamente, ma senza ridondanze, la funzione in oggetto. Se la rete è troppo ricca andremo in overfitting.

In pratica, per ottenere una buona generalizzazione, si fa l’apprendimento su un set di dati, quello degli esempi \mathcal{E} , ma la verifica sul funzionamento viene fatta su un insieme diverso, quello di test \mathcal{T} come schematizzato in figura 27. Si osserva che, durante l’apprendimento, l’errore sull’insieme degli esempi diminuisce sempre al crescere delle iterazioni mentre l’errore sull’insieme di test tende a raggiungere un minimo per poi risalire. Si attribuisce la risalita dell’errore sull’insieme di test all’overfitting sull’insieme degli esempi. Il momento migliore per fermare l’apprendimento è quando si raggiunge il minimo dell’errore sul set di test e in queste condizioni la rete ha la migliore generalizzazione, si veda un esempio in figura 28.

È molto importante avere delle relazioni quantitative, per prima cosa osserviamo che la funzione appresa f ripartisce sempre l’insieme dei 2^n possibili input in due sottinsiemi e possiamo pensare la generalizzazione come l’overlap fra questa ripartizione e quella indotta dalla funzione *vera* f_v . Per

definire meglio il concetto di generalizzazione si possono definire e studiare varie quantità come per esempio:

- il numero medio di generalizzazioni diverse possibili a partire dal set degli esempi \mathcal{E} ;
- la probabilità *media* P che la rete dia una risposta corretta per un dato input. La media si intende fatta su tutte le possibili reti che hanno appreso \mathcal{E} ;
- la probabilità P che la rete dia una risposta corretta per un dato input nel caso *peggiore* per tutte le possibili reti che hanno appreso \mathcal{E} .

Chiaramente quest'ultimo valore dà un limite superiore alla probabilità di errore e ci concentriamo su questo caso. La teoria che segue si applica non solo alle reti neurali ma a vari tipi di approssimazione.

Definiamo formalmente la generalizzazione come un *funzionale* $g(f)$ che dà la probabilità P che $f(\vec{x}) = f_v(\vec{x})$ su un esempio casuale \vec{x} . Si noti che $g(f)$ misura quanto bene f approssima f_v ed è indipendente dall'insieme degli esempi \mathcal{E} ma chiaramente sarà assai difficile da calcolare. Nel caso semplice ma irrealistico di \vec{x} tutti equiprobabili chiaramente la generalizzazione si riduce a:

$$g(f) = \frac{|\{\vec{x} : f(\vec{x}) = f_v(\vec{x})\}|}{|\{\vec{x}\}|}$$

dove $|\{\cdot\}|$ indica la cardinalità dell'insieme e, nel nostro caso, $|\{\vec{x}\}| = 2^n$. In generale se potessimo conoscere $g(f)$ allora potremmo usarla come criterio di stop per l'apprendimento per esempio quando raggiunge il valore di 0.95.

Sia ora $g_m(f)$ lo stesso funzionale calcolato però solo sull'insieme di m esempi \mathcal{E} . Di solito si riesce a ottenere che gli esempi siano imparati perfettamente, cioè $g_m(f) = m/m = 1$. Per la legge dei grandi numeri è lecito aspettarsi che $\lim_{m \rightarrow \infty} g_m(f) = g(f)$ però per m finiti in generale $g_m(f)$ sarà maggiore di $g(f)$ e dunque non è una buona stima della generalizzazione vera $g(f)$. Per esempio nel caso di una rete che realizza la “grandmother cell representation” si ha $g_m(f) = 1$ ma $g(f) = m/2^n \approx 0$.

Nel 1971 Vapnik e Cervonenkis sono riusciti a dimostrare che:

$$P \left(\max_f |g_m(f) - g(f)| > \varepsilon \right) \leq 4G(2m)e^{-m\frac{\varepsilon^2}{8}} \quad (18)$$

dove $G(m)$ è la growth function, già introdotta nel caso del Perceptron. Chiaramente il termine $e^{-m\frac{\varepsilon^2}{8}}$ fa andare a 0 il secondo termine per $m \rightarrow \infty$ ¹⁸. Con un linguaggio più preciso diciamo che la (18) mostra la convergenza

¹⁸purché $G(m) \leq 2^m$ cresca meno che esponenzialmente al crescere di m e nel contrasto fra questi due termini si gioca la partita fra numero di parametri/pesi, che fanno crescere $G(m)$ e numero di esempi m che fanno andare a zero e^{-m} .

uniforme della frazione di risposte corrette su \mathcal{E} , $g_m(f)$, alla probabilità di generalizzazione $g(f)$.

Prima di vedere come applicare questo risultato osserviamo che nel caso più generale assumeremo che ognuno dei 2^n input possibili \vec{x} abbia una certa probabilità $\mathcal{P}(\vec{x})$ di verificarsi, in altre parole assumeremo una distribuzione di probabilità degli input. Finora avevamo implicitamente assunto che tutti gli input fossero equiprobabili ($\mathcal{P}(\vec{x}) = 2^{-n}$) ma chiaramente questo non è il caso più generale. Per vederlo nel nostro esempio delle ‘a’ è ragionevole assumere che una a scritta in caratteri gotici, appaia meno di frequente che una normale a stampata, e, di conseguenza, un errore nel riconoscere una a ‘normale’, e dunque più frequente, sia più grave che un errore nel riconoscere una a scritta in gotico. Un altro esempio può venire dalle immagini, ritornando al nostro piccolo display di 10×10 pixel scegliendo a caso fra le 10^{30} immagini possibili troveremo solo immagini che definiremmo di puro “rumore”, le immagini che noi giudichiamo tali, le immagini “naturali”, sono un piccolissimo sottinsieme delle immagini possibili e con una distribuzione molto particolare (della quale accenneremo nel paragrafo 13.5.2). Insomma la definizione più corretta di $g(f)$ terrà conto di questo pesando i diversi input \vec{x} con $\mathcal{P}(\vec{x})$.

Vediamo con un esempio come si può usare la (18): scegliendo un numero di esempi m sufficientemente grande possiamo portare il limite superiore a δ (per esempio $\delta = 0.01$) e se, contemporaneamente, siamo riusciti ad apprendere perfettamente il nostro set di esempi avremo $g_m(f) = 1$ ¹⁹. In questo caso possiamo dedurre che con probabilità

$$P\left(\max_f 1 - g(f) > \varepsilon\right) = P\left(\min_f g(f) < 1 - \varepsilon\right) \leq \delta$$

dalla quale, considerando l’evento opposto,

$$P\left(\min_f g(f) > 1 - \varepsilon\right) \geq 1 - \delta = 0.99$$

in altre parole la generalizzazione della nostra rete è, con probabilità maggiore o uguale al 99%,

$$g(f) > 1 - \varepsilon$$

in virtù del fatto che la formula dà la generalizzazione per la peggiore fra tutte le possibili f e per cui, a fortiori, anche per la particolare f che la nostra rete sta implementando.

Possiamo calcolare il limite superiore della (18) se conosciamo $G(m)$ oppure un suo limite superiore, per esempio quello (17); in questo caso basterà conoscere d_{VC} . Se esaminiamo il caso del Perceptron avremo dunque

¹⁹il che presuppone che il problema descritto dagli m esempi sia risolubile dalla rete considerata, per esempio il Perceptron.

che $G(m) \leq m^n + 1$ e dunque il limite superiore della (18) sarà $4[(2m)^n + 1]e^{-m\frac{\varepsilon^2}{8}}$. Se chiediamo che questo limite superiore sia a sua volta maggiorato dalla costante δ troviamo

$$4[(2m)^n + 1]e^{-m\frac{\varepsilon^2}{8}} \leq \delta$$

che diventa

$$e^{\log[(2m)^n + 1]}e^{-m\frac{\varepsilon^2}{8}} \leq \frac{\delta}{4}$$

e dunque

$$\log[(2m)^n + 1] - m\frac{\varepsilon^2}{8} \leq \log \frac{\delta}{4}$$

se assumiamo che n e m siano grandi possiamo trascurare il $+1$ nel logaritmo e restiamo con

$$n \log(2m) - m\frac{\varepsilon^2}{8} \leq \log \frac{\delta}{4}$$

cioè

$$m \geq \frac{8}{\varepsilon^2} [n \log(2m) - \log \frac{\delta}{4}]$$

per valutare come si comporta questo limite per $n, m \rightarrow \infty$ con $m/n \rightarrow \alpha =$ costante vediamo che l'ultima disegualanza da

$$m \geq \frac{8}{\varepsilon^2} O(n \log n)$$

che indica che gli esempi necessari per l'apprendimento del Perceptron cresce solo con $n \log n$ del numero degli input, un numero addirittura minore di n^2 e comunque di gran lunga inferiore al numero di input possibili 2^n ; dunque sembrerebbe un bel successo. Attenzione però che per piccoli ε , il fattore $\frac{8}{\varepsilon^2}$ può essere dominante.

Per altre reti (per esempio quelle feed-forward) non è possibile calcolare d_{VC} ma solo, talvolta, un limite superiore. In particolare per una rete con N neuroni e con un totale di W pesi è stato dimostrato che

$$d_{VC} \leq 2W \log_2(eN)$$

(e è la costante numerica) questa disegualanza può essere usata secondo la stessa procedura usata per il Perceptron per dimostrare che, se l'errore sull'insieme di esempi \mathcal{E} è minore di $\frac{\varepsilon}{2}$, occorrono almeno $\frac{W}{\varepsilon} \log \frac{N}{\varepsilon}$ esempi per ottenere una generalizzazione migliore di $1 - \varepsilon$.

Questa relazione mostra una dipendenza, secondo la funzione $x \log x$ dalla variabile $\frac{W}{\varepsilon}$ che è intuitiva: il numero di esempi necessari cresce al crescere del numero dei pesi ed al diminuire dell'errore di generalizzazione che si è disposti a tollerare.

9 Apprendimento non supervisionato

Abbiamo finora parlato di reti neurali nelle quali l'apprendimento veniva fatto dal confronto fra l'output reale e quello desiderato, per cui era come se fosse presente un insegnante o supervisore, che diceva alla rete quando l'output era sbagliato e, in questo caso, correggeva i pesi. Vedremo ora come si possa fare dell'apprendimento anche senza il supervisore, semplicemente dando alla rete una regola per aggiornare i pesi in base all'input. Prima di addentrarci alcune considerazioni generali sull'apprendimento come l'abbiamo visto finora:

- possiamo pensare ad una rete che ha completato l'apprendimento come ad un *algoritmo-programma* che trasforma l'input in output;
- il programma prodotto dall'apprendimento "gira" in parallelo sui singoli neuroni e, spesso, è in grado di produrre risultati non banali. In altre parole non sarebbe facile scrivere un programma standard, per esempio in C, che sia in grado di ottenere gli stessi risultati;
- questo programma scritto nella rete nella maggior parte dei casi non è "comprensibile" nel senso che non siamo in grado, guardando i valori dei pesi, di capire "come" la rete ottiene i risultati. Dai valori dei pesi non siamo in grado di scrivere un programma di tipo tradizionale che implementi la stessa funzione;
- il programma ottenuto è "adattivo" nel senso che quando l'apprendimento è avvenuto siamo ad un minimo della funzione E e, in media, $\Delta w_i = 0$. Se però il problema si modifica l'apprendimento si rimette in moto variando i pesi per adattarsi alla nuova situazione: esempio filtri adattivi;
- comunque, in genere, la back-propagation è un processo di apprendimento incredibilmente lento.

Finora abbiamo sempre supposto che l'apprendimento avvenisse grazie ad una supervisione esterna: ora abbandoniamo quest'idea. Per procedere avremo però bisogno di un'altra guida che sarà la distribuzione degli input. Alla fine dello scorso capitolo abbiamo dovuto introdurre l'idea che gli input \vec{x} non siano in generale tutti equiprobabili ma seguano una distribuzione $\mathcal{P}(\vec{x})$. Questo concetto farà da filo conduttore per questo capitolo e avrà conseguenze profonde.

9.1 Un esempio iniziale

Iniziamo l'apprendimento non supervisionato con un problema semplice: costruire una rete che riconosca la "familiarità" di un input. In altre parole

dato un insieme di esempi $\mathcal{E} = \{\vec{x}_\nu : \nu = 1, \dots, m\}$ ed una distribuzione associata $\mathcal{P}(\vec{x})$ vogliamo una rete che dia un valore in uscita che sia maggiore, maggiore è la probabilità (o frequenza, se preferite) dell'input presentato all'ingresso. Un esempio potrebbe essere quello di una rete che riconosca (quello che noi definiremmo) un immagine nel mare magnum delle immagini casuali.

Questa idea ricorda quanto visto nel capitolo 8 e rappresentato graficamente in figura 25 quando eravamo alla ricerca di un insieme di esempi che fosse rappresentativo di tutti i dati appartenenti a quella classe solamente che in quel caso volevamo definire i “confini” dei nostri esempi (per poi dire: quello che è dentro è buono, quello che è fuori è cattivo) ora invece cerchiamo un'altra rappresentazione degli esempi in termini di un output che sia simile a $\mathcal{P}(\vec{x})$. Si può vederlo come un raffinamento dell'idea precedente che implicitamente assumeva $\mathcal{P} = 1$ dentro il confine dei dati giusti e 0 fuori.

Iniziamo ad esaminare un caso molto semplice di una rete formata da un solo neurone con funzione di trasferimento lineare per cui si avrà che l'uscita sarà data da $y_\nu = \vec{x}_\nu \cdot \vec{w} := \vec{x}_\nu^T \vec{w}$. Visto che in questi paragrafi useremo la notazione matriciale usiamo anche il simbolo \vec{x}_ν^T per indicare il vettore riga trasposto di \vec{x}_ν . A questa rete applichiamo un algoritmo di apprendimento $w_i = w_i + \varepsilon \Delta w_i$ con un Δw_i Hebbiano

$$\Delta w_i = y x_i \tag{19}$$

per cui la regola di apprendimento, in forma vettoriale, sarà

$$\vec{w} = \vec{w} + \varepsilon y \vec{x}$$

Iniziamo calcolando il valore medio di Δw_i che si otterebbe dopo un ipotetico passaggio di tutti gli input possibili. Questa media su tutti gli input apparirà così di frequente nel seguito che, per alleggerire la notazione, introduciamo una notazione ad hoc: $\langle \Delta w_i \rangle$. Facilmente dalla (19) si trova

$$\begin{aligned} \langle \Delta w_i \rangle &:= \frac{1}{m} \sum_{\nu=1}^m \Delta w_{i\nu} = \frac{1}{m} \sum_{\nu=1}^m y_\nu x_{i\nu} = \frac{1}{m} \sum_{\nu=1}^m \sum_{j=1}^n x_{j\nu} w_j x_{i\nu} = \\ &= \frac{1}{m} \sum_{j=1}^n w_j \sum_{\nu=1}^m x_{i\nu} x_{j\nu} . \end{aligned}$$

Definendo la matrice C di elementi

$$C_{ij} := \frac{1}{m} \sum_{\nu=1}^m x_{i\nu} x_{j\nu} = \langle x_i x_j \rangle$$

che è, in sostanza, per dati a media nulla $\langle x_i \rangle = 0$, la matrice di *covarianza* degli input (vedi appendice A.3), possiamo scrivere

$$\langle \Delta w_i \rangle = \sum_{j=1}^n C_{ij} w_j$$

che, in forma matriciale, diventa

$$\langle \Delta \vec{w} \rangle = C \vec{w}$$

e, approssimando la variazione dei pesi con la *variazione media*, il nuovo \vec{w} , dopo un'applicazione dell'algoritmo di apprendimento, varrà

$$\vec{w} = \vec{w} + \varepsilon \langle \Delta \vec{w} \rangle = \vec{w} + \varepsilon C \vec{w} = (\mathbb{1} + \varepsilon C) \vec{w}$$

dove $\mathbb{1}$ indica la matrice identità, e dopo s applicazioni

$$\vec{w} = (\mathbb{1} + \varepsilon C)^s \vec{w} .$$

Scrivendo \vec{w} nella base degli autovettori \vec{e}_i della matrice C , assume la forma $\vec{w} = \sum_{i=1}^n \alpha_i \vec{e}_i$ (con $\alpha_i = \vec{e}_i^T \vec{w}$, dimostratelo) e dopo un'applicazione avremo

$$\vec{w} = \vec{w} + \varepsilon C \vec{w} = \sum_{i=1}^n \alpha_i \vec{e}_i + \varepsilon C \sum_{i=1}^n \alpha_i \vec{e}_i = \sum_{i=1}^n \alpha_i \vec{e}_i + \varepsilon \sum_{i=1}^n \alpha_i \lambda_i \vec{e}_i = \sum_{i=1}^n (1 + \varepsilon \lambda_i) \alpha_i \vec{e}_i$$

e, dopo s applicazioni della regola di apprendimento,

$$\vec{w} = \sum_{i=1}^n (1 + \varepsilon \lambda_i)^s \alpha_i \vec{e}_i$$

la quale ci dice che, essendo $\lambda_i \geq 0$ ossia $1 + \varepsilon \lambda_i \geq 1$, in media, al crescere delle iterazioni, il vettore \vec{w} tenderà a crescere indefinitamente orientandosi allo stesso tempo nella direzione dell'autovettore corrispondente all'autovalore massimo.

Ora capiamo perché abbiamo detto che questo tipo di rete misura la familiarità dei dati: dato il vettore dei pesi tende all'autovettore dell'autovalore massimo, in sostanza l'output y rappresenta la proiezione dei dati lungo la direzione della prima *componente principale* dei dati (vedi appendice A.4). In un certo senso, è la miglior sintesi dei dati che si può avere in uno spazio unidimensionale.

9.2 La regola di Oja

Nel 1982 Erkki Oja propose una regola di apprendimento più raffinata della (19)

$$\Delta w_i = y x_i - y^2 w_i = y(x_i - y w_i)$$

che in forma vettoriale diventa

$$\Delta \vec{w} = y(\vec{x} - y \vec{w}) \tag{20}$$

che si può interpretare a parole dicendo che la variazione dei pesi è data dall'output moltiplicato per l'input al quale si sottrae l'output propagato indietro, una specie di feedback. Con questa regola di apprendimento possiamo calcolare nuovamente

$$\begin{aligned}\langle \Delta w_i \rangle &= \langle yx_i \rangle - \langle y^2 w_i \rangle = \left\langle \sum_{j=1}^n w_j x_j x_i \right\rangle - \left\langle \sum_{k=1}^n \sum_{l=1}^n w_k x_k w_l x_l w_i \right\rangle = \\ &= \sum_{j=1}^n \langle x_j x_i \rangle w_j - \sum_{k=1}^n \sum_{l=1}^n w_k \langle x_k x_l \rangle w_l w_i = \\ &= \sum_{j=1}^n C_{ij} w_j - \sum_{k=1}^n \sum_{l=1}^n w_k C_{kl} w_l w_i\end{aligned}$$

che, in forma vettoriale diventa (facile da ricavare anche direttamente in forma vettoriale, fatelo per esercizio)

$$\langle \Delta \vec{w} \rangle = C \vec{w} - \vec{w}^T C \vec{w} \vec{w} . \quad (21)$$

In questo caso vediamo che il vettore dei pesi non necessariamente cresce indefinitamente a causa della riduzione data dal secondo termine, perciò si può intuire che tenderà a una posizione di equilibrio. Una condizione necessaria per l'equilibrio è che \vec{w} in media non cambi e cioè $\langle \Delta \vec{w} \rangle = 0$. Questa richiesta è soddisfatta se

$$C \vec{w} = \vec{w}^T C \vec{w} \vec{w}$$

che significa che, all'equilibrio, \vec{w} è un autovettore di C (dato che $\vec{w}^T C \vec{w}$ è una quantità scalare) di autovalore $\vec{w}^T C \vec{w} := \lambda$. Alla stabilità avremo

$$\lambda = \vec{w}^T C \vec{w} = \lambda \vec{w}^T \vec{w}$$

dalla quale segue che $\vec{w}^T \vec{w} = 1$, cioè alla stabilità \vec{w} deve essere un autovettore normalizzato. Dunque la regola (20) ha come punti di equilibrio $\vec{w} = \vec{e}_i$, gli autovettori della matrice di covarianza C per di più normalizzati.

Il valore medio dell'output quadrato (che coincide con la varianza se il valore medio dell'output è nullo) vale:

$$\langle y^2 \rangle = \langle (\vec{w}^T \vec{x})^2 \rangle = \langle \vec{w}^T \vec{x} \vec{x}^T \vec{w} \rangle = \vec{w}^T C \vec{w}$$

per cui alla stabilità avremo $\vec{w} = \vec{e}_i$ e $\langle y^2 \rangle = \lambda_i$, l'autovalore corrispondente. Per le proprietà di autovalori e autovettori sappiamo che

$$\lambda_{min} \leq \frac{\vec{w}^T C \vec{w}}{\vec{w}^T \vec{w}} \leq \lambda_{max}$$

per cui il massimo valore che può assumere $\langle y^2 \rangle$ alla stabilità è λ_{max} .

Mostriamo ora che, pur essendo tutti gli autovettori delle soluzioni di equilibrio, solo l'autovettore corrispondente all'autovalore massimo è una soluzione di equilibrio stabile. Sia inizialmente $\vec{w} = \vec{e}_i$ e supponiamo di perturbarlo di una quantità infinitesima η nella direzione di un altro autovettore, cioè $\vec{w} = \vec{e}_i + \eta\vec{e}_j$. Calcoliamo $\langle \Delta\vec{w} \rangle$, si trova

$$\langle \Delta\vec{w} \rangle = C\vec{w} - \vec{w}^T C \vec{w} \vec{w} = C(\vec{e}_i + \eta\vec{e}_j) - [(\vec{e}_i + \eta\vec{e}_j)^T C (\vec{e}_i + \eta\vec{e}_j)] (\vec{e}_i + \eta\vec{e}_j)$$

e per sviluppare il secondo termine ricordiamo che C è simmetrica per cui $\vec{e}_i^T C = \lambda_i \vec{e}_i^T$. Così troviamo

$$\langle \Delta\vec{w} \rangle = \lambda_i \vec{e}_i + \eta \lambda_j \vec{e}_j - (\lambda_i + \eta^2 \lambda_j)(\vec{e}_i + \eta\vec{e}_j) = \eta(\lambda_j - \lambda_i)\vec{e}_j$$

dove abbiamo tenuto solo i termini al primo ordine in η . Dunque in un ciclo di apprendimento $\vec{w} = \vec{w} + \epsilon \langle \Delta\vec{w} \rangle$ avremo

$$\vec{e}_i + \eta\vec{e}_j = \vec{e}_i + \eta\vec{e}_j + \epsilon \eta(\lambda_j - \lambda_i)\vec{e}_j = \vec{e}_i + \eta[1 + \epsilon(\lambda_j - \lambda_i)]\vec{e}_j$$

dunque \vec{e}_i resta stabile (c'era da aspettarselo) mentre la perturbazione nella direzione di \vec{e}_j è stata moltiplicata per un fattore $[1 + \epsilon(\lambda_j - \lambda_i)]$ e dopo s passi di apprendimento la perturbazione sarà moltiplicata per $[1 + \epsilon(\lambda_j - \lambda_i)]^s$. Dunque se $\lambda_j > \lambda_i$, in media, la componente della perturbazione cresce esponenzialmente, indicando che il vettore \vec{w} è attirato verso questa direzione. Questo processo varrà fintantoché non ci si sarà stabilizzati sull'autovettore dell'autovalore massimo quando qualsiasi perturbazione tenderà a riportarlo verso la condizione di equilibrio (provate a dimostrarlo).

Riassumendo abbiamo dimostrato che la regola di apprendimento (20)

- rende i vari autovettori di C punti di equilibrio e normalizza il vettore dei pesi \vec{w} ;
- fa convergere \vec{w} , in media, verso l'autovettore di C corrispondente all'autovalore massimo;
- massimizza $\langle y^2 \rangle$.

in altre parole da un neurone lineare che esegua questo algoritmo si ricava sia l'autovettore corrispondente all'autovalore massimo di C (il vettore dei pesi), che l'autovalore stesso ($\langle y^2 \rangle$).

È ovvio che un neurone di questo tipo è ideale per calcolare, con una rete neurale, le componenti principali dei dati, metodo descritto per sommi capi nell'appendice A.4.

9.3 La regola di Sanger

Nel 1989 Sanger propone una regola di apprendimento per una rete con n neuroni di ingresso e n neuroni nello strato successivo il cui i -esimo output

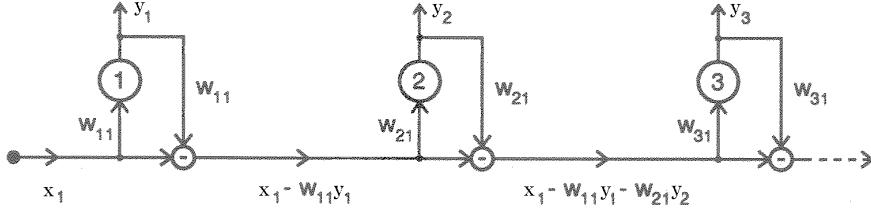


Figura 29: Rete che implementa la regola di Sanger, è rappresentato solo l'input x_1 , per gli altri input ci sono linee parallele e identiche.

vale $y_i = \vec{x}^T \vec{w}_i = \sum_{j=1}^n x_j w_{ij}$ data da

$$\Delta w_{ij} = y_i(x_j - \sum_{k=1}^i y_k w_{kj})$$

che in forma vettoriale diventa

$$\Delta \vec{w}_i = y_i(\vec{x} - \sum_{k=1}^i y_k \vec{w}_k) \quad (22)$$

come si vede la regola non è più locale in quanto per aggiornare i pesi di un neurone devo avere gli output e i pesi di tutti i neuroni precedenti. Dal punto di vista formale questo problema si può in parte ovviare riscrivendo la regola di apprendimento

$$\Delta \vec{w}_i = y_i(\vec{x} - \sum_{k=1}^{i-1} y_k \vec{w}_k - y_i \vec{w}_i)$$

e considerando dei neuroni in cascata in maniera che l'input del i -esimo neurone sia $\vec{x} - \sum_{k=1}^{i-1} y_k \vec{w}_k$: ogni neurone sottrae il suo output prima di passare avanti l'informazione. In questo modo tutti i neuroni della cascata apprendono con la regola di Oja, il principio è schematizzato in figura 29.

Al solito calcoliamo la variazione media di un peso per trovare i punti di equilibrio della rete

$$\langle \Delta w_{ij} \rangle = \langle y_i x_j \rangle - \langle \sum_{k=1}^i y_k y_k w_{kj} \rangle = \sum_{l=1}^n C_{jl} w_{il} - \langle \sum_{k=1}^i \sum_{q=1}^n \sum_{r=1}^n w_{kq} x_q x_r w_{ir} w_{kj} \rangle$$

e il secondo termine da $\sum_{k=1}^i (\vec{w}_k^T C \vec{w}_i) w_{kj}$ per cui, in forma vettoriale, si ha

$$\langle \Delta \vec{w}_i \rangle = C \vec{w}_i - \sum_{k=1}^i (\vec{w}_k^T C \vec{w}_i) \vec{w}_k$$

Vogliamo ora dimostrare che se ci sono n neuroni nel secondo strato essi hanno posizioni di equilibrio nelle rispettive componenti principali dei dati.

Se esaminiamo la relazione appena trovata per il primo neurone del secondo strato vediamo che essa ricalca la condizione di equilibrio per la regola di Oja (21) per la quale abbiamo già dimostrato che l'equilibrio stabile si ha al primo autovettore della matrice di covarianza. Per completare la dimostrazione procediamo per induzione e supposto che i primi $i - 1$ neuroni abbiano già posizioni di equilibrio ai primi $i - 1$ autovettori \vec{e} della matrice di covarianza troviamo la posizione di equilibrio dell' i -esimo neurone. Riscriviamo la formula precedente come

$$\langle \Delta \vec{w}_i \rangle = C \vec{w}_i - \sum_{k=1}^{i-1} (\vec{w}_k^T C \vec{w}_i) \vec{w}_k - (\vec{w}_i^T C \vec{w}_i) \vec{w}_i$$

e, dato che i primi $i - 1$ vettori \vec{w} coincidono con gli autovettori di C , essi formano un sistema ortonormale di $i - 1$ assi. Di conseguenza $C \vec{w}_i - \sum_{k=1}^{i-1} (\vec{w}_k^T C \vec{w}_i) \vec{w}_k$ può essere interpretato come il vettore $C \vec{w}_i$ cui siano state sottratte tutte le sue $i - 1$ proiezioni nello spazio generato da $\vec{w}_1 \dots \vec{w}_{i-1}$. Di conseguenza quello che rimane sono le componenti di $C \vec{w}_i$ nello spazio ortogonale all'iperpiano di $\vec{w}_1 \dots \vec{w}_{i-1}$. Indicando questo vettore con $(C \vec{w}_i)^\perp$ abbiamo

$$\langle \Delta \vec{w}_i \rangle = (C \vec{w}_i)^\perp - (\vec{w}_i^T C \vec{w}_i) \vec{w}_i$$

Siccome C lascia invariante lo spazio dei suoi autovettori ne segue che $(C \vec{w}_i)^\perp = C \vec{w}_i^\perp$. Supponiamo ora che \vec{w}_i abbia delle componenti nello spazio di $\vec{w}_1 \dots \vec{w}_{i-1}$, chiamiamole \vec{w}_i^* , allora per esse la relazione appena trovata diventa $\langle \Delta \vec{w}_i^* \rangle = 0 - (\vec{w}_i^{*T} C \vec{w}_i^*) \vec{w}_i^*$ che mostra, dato che $\vec{w}_i^{*T} C \vec{w}_i^* \geq 0$, che per queste componenti si avrà un decadimento esponenziale e dunque in breve \vec{w}_i diventa \vec{w}_i^\perp per il quale possiamo scrivere

$$\langle \Delta \vec{w}_i^\perp \rangle = C \vec{w}_i^\perp - (\vec{w}_i^{\perp T} C \vec{w}_i^\perp) \vec{w}_i^\perp$$

Questa relazione altro non è che la regola di Oja (21) e dunque sappiamo che la sua posizione di equilibrio corrisponde all'autovettore \vec{e}_i di autovettore massimo λ_i in questo sottospazio, col che abbiamo dimostrato quanto volevamo.

Osserviamo che, sia per la regola di Oja che per quella di Sanger, abbiamo solo dimostrato le condizioni di equilibrio e non abbiamo affrontato le proprietà di convergenza della rete che dipendono, fra le altre cose, da ϵ e dal numero di iterazioni. Alcuni punti critici che andrebbero affrontati per completare l'analisi sono:

- sotto quali condizioni (per esempio su ϵ , sulla distribuzione degli esempi etc. etc.) le posizioni di equilibrio possano essere raggiunte e/o mantenute ? Esistono attrattori ciclici oltre che stazionari ?

- i neuroni della rete di Sanger evolvono tutti contemporaneamente e non uno alla volta come implicitamente assunto nella dimostrazione. La nostra dimostrazione di equilibrio vale lo stesso ? (risposta: si);
- con ε finito anche dopo molti cicli di apprendimento \vec{w} si muoverà, in modo più o meno periodico, attorno al punto di equilibrio, cosa si può fare per stabilizzarlo ? (risposta: decrescere ε).

Rispondere a queste e altre domande è compito della teoria dei sistemi dinamici stocastici ma questo esula dal contenuto di questo corso.

Per concludere due osservazioni: la prima è che la regola di Sanger si può anche derivare da una rete di 3 strati di neuroni lineari con, rispettivamente, $n, h(\leq n)$ e n neuroni, con valori x_i, y_i e z_i e con una funzione d'errore nella quale il valore desiderato per ogni output è il valore del corrispondente input e cioè:

$$E = \frac{1}{2} \sum_{\nu=1}^m \sum_{i=1}^n (z_{i\nu} - x_{i\nu})^2 .$$

In questo caso la back-propagation, ovverosia la discesa lungo il gradiente di E , genera la regola di Sanger e si trova che gli h neuroni dello strato intermedio convergono alle prime h componenti principali degli input. Infatti è una proprietà delle componenti principali di fornire la miglior approssimazione dei dati in un sottospazio a $h(\leq n)$ dimensioni (vedi appendice A.4). Una rete di questo genere può essere usata per fare una semplice, ma relativamente efficiente, compressione di dati, per esempio di immagini.

Osserviamo che la rete di Sanger calcola le componenti principali, una tecnica ben nota ormai da più di un secolo, per cui non aggiunge niente alla nostra conoscenza. Come tutti i metodi basati sull'apprendimento però ha il vantaggio di essere *adattiva*: se cambia la distribuzione dei dati si adatteranno anche le componenti principali senza bisogno né di rilevare il cambiamento né di ricalcolare le componenti principali. Questo non piccolo vantaggio sui metodi tradizionali viene sfruttato, per esempio, in molti filtri adattivi.

La limitazione principale di questo tipo di reti è che operano trasformazioni lineari dei dati con tutte le semplificazioni e limitazioni che questo comporta. Per esempio non ha senso aggiungere strati di reti di questo tipo dato che, essendo lineari, l'effetto di più strati può sempre essere ridotto ad un unico strato equivalente. Per procedere avremo bisogno di reintrodurre neuroni con funzione di trasferimento non lineare.

9.4 Le reti con l'apprendimento competitivo

Un primo esempio di reti non supervisionate con funzioni di trasferimento non lineare è quello con un architettura simile a quelle delle sezioni precedenti ma nel quale si richiede che uno solo dei neuroni dello strato finale

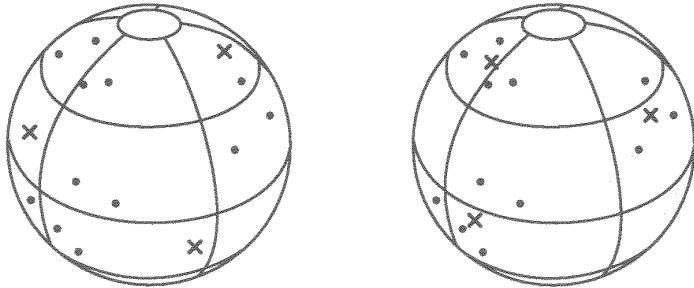


Figura 30: Prima e dopo l'apprendimento competitivo: i punti rappresentano gli esempi e le croci i vettori dei pesi.

sia attivo, da qui l'idea di competizione: solo il neurone vincente è attivo e apprende. In inglese queste reti sono chiamate “Winner Take All” (WTA) e il loro uso tipico è per categorizzare i dati: dato un input il neurone attivo indica a quale categoria l'input appartiene, questo processo viene detto “vector quantization”: per esempio potrebbe trattarsi di una rete per riconoscere le solite lettere.

I neuroni del secondo strato darebbero output $y_i = \text{sgn}(\vec{x} \cdot \vec{w}_i)$ ma il meccanismo WTA fa sì che solo il neurone “vincitore”, quello con il maggior valore di $\vec{x} \cdot \vec{w}_i$, produca output positivo e sia l'unico al quale si applica l'algoritmo di apprendimento; tutti gli altri neuroni del secondo strato danno output negativo. Dal punto di vista pratico si può rendere attivo solo il neurone vincitore aggiungendo un altro strato oppure introducendo delle connessioni di inibizione laterale fra i vari neuroni, quest'ultimo caso ha maggior verosimiglianza biologica. Nel caso le reti siano simulate basterà un semplice loop sui vari outputs.

Da quanto detto è chiaro che il neurone i^* risulta il vincitore se

$$\vec{x} \cdot \vec{w}_{i^*} > \vec{x} \cdot \vec{w}_j \quad \forall j \neq i^*$$

Osserviamo che, se i pesi e gli input sono normalizzati (o comunque hanno lunghezza uguale), il prodotto scalare coincide con il coseno dell'angolo fra i due vettori e possiamo equivalentemente definire il neurone vincitore come quello per il quale è minima la distanza fra il suo vettore dei pesi e \vec{x} e cioè

$$|\vec{w}_{i^*} - \vec{x}| < |\vec{w}_j - \vec{x}| \quad \forall j \neq i^*$$

(provate a dimostrarlo per esercizio).

Possiamo rappresentare sia gli input che i pesi nello stesso spazio e se i pesi sono normalizzati il neurone vincitore è quello che è maggiormente “vicino” all'input. In questo modo i neuroni ripartiscono lo spazio degli input in vari insiemi di punti che sono quelli che hanno minor distanza da un dato neurone: questi insiemi formano una “tassellazione di Voronoi” dello

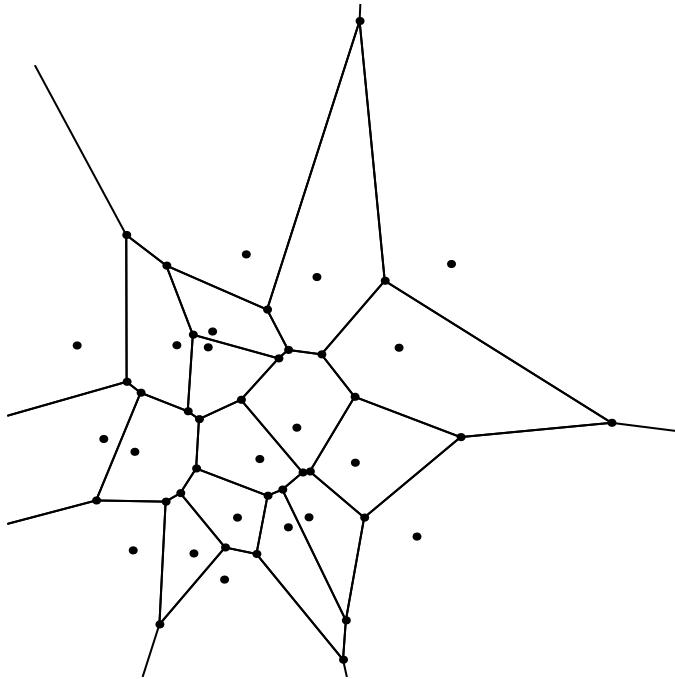


Figura 31: Tassellazione di Voronoi per $m = 20$ punti nello spazio a $n = 2$ dimensioni; si noti che le celle sono convesse.

spazio degli input e la figura 31 da un esempio di tassellazione di Voronoi bidimensionale.

In questa rete solo il neurone vincitore apprende e, data la solita regola,

$$\vec{w}_i = \vec{w}_i + \varepsilon \Delta \vec{w}_i$$

si può definire per esempio

$$\Delta \vec{w}_{i^*} = \vec{x}$$

in questo modo il neurone vincitore si avvicina all'input che lo ha fatto vincere e alla prossima occasione sarà ancora più vicino e questo input. In realtà questa regola farebbe crescere i pesi indefinitamente e bisogna trovare un modo di normalizzare i pesi. Questo si può ottenere con la regola

$$\Delta \vec{w}_{i^*} = \vec{x} - \vec{w}_{i^*}$$

che vale se pesi e input sono normalizzati. Possiamo infatti calcolare la variazione del vettore \vec{w} generata dall'apprendimento come:

$$[\vec{w} + \varepsilon(\vec{x} - \vec{w})]^2 - \vec{w}^2 = 2\varepsilon\vec{w} \cdot (\vec{x} - \vec{w}) + \varepsilon^2(\vec{x} - \vec{w})^2 \simeq 2\varepsilon\vec{w} \cdot (\vec{x} - \vec{w})$$

dove abbiamo trascurato i termini con ε^2 . Osserviamo ora che se viene applicato l'apprendimento il neurone con pesi \vec{w} avrà vinto la competizione

e sarà il più vicino a \vec{x} per cui, visto che abbiamo supposto che $\vec{x}^2 = \vec{w}^2 = 1$ si avrà $\vec{w} \cdot (\vec{x} - \vec{w}) \simeq 0$ e il modulo di \vec{w} non varia nell'apprendimento, almeno al primo ordine in ε ; la situazione è rappresentata in figura 30.

A questa regola di apprendimento si può associare una funzione dei pesi del tipo

$$E(W) = \frac{1}{2} \sum_{\nu=1}^m |\vec{x}_\nu - \vec{w}_{i^*(\nu)}|^2 = \frac{1}{2} \sum_{\nu=1}^m \sum_{j=1}^h |\vec{x}_\nu - \vec{w}_j|^2 M_{j\nu}$$

dove, dato che per ogni input \vec{x}_ν abbiamo un neurone “vincitore” \vec{w}_{i^*} per rendere esplicita questa dipendenza lo abbiamo indicato con $i^*(\nu)$ e dove W indica l'insieme di tutti gli h vettori dei pesi \vec{w}_j . In questo modo nella prima sommatoria si sommeranno solo le distanze fra un neurone e gli input che appartengono alla cella di Voronoi che lui induce. Nella seconda uguaglianza per descrivere la funzione $i^*(\nu)$ abbiamo introdotto una matrice $M_{j\nu}$ che, per ogni input, vale 1 solo per il neurone vincitore e 0 per tutti gli altri. Osserviamo che questa matrice rende la funzione $E(W)$ continua solo a tratti e, per di più, siccome i pesi variano, essa si modifica durante l'apprendimento e di conseguenza anche la funzione d'errore $E(W)$ si modificherà durante l'apprendimento, una caratteristica mai riscontrata finora. Scriviamo ora la funzione $E(W)$ in funzione delle singole componenti

$$E(W) = \frac{1}{2} \sum_{\nu=1}^m \sum_{j=1}^h \sum_{i=1}^n (x_{i\nu} - w_{ij})^2 M_{j\nu}$$

e facilmente calcoliamo

$$\frac{\partial E}{\partial w_{lk}} = - \sum_{\nu(k)} (x_{l\nu} - w_{lk})$$

che risulta essere la somma dei termini della somma su tutti gli input $\nu(k)$ che rendono il neurone k il vincitore. Se scriviamo il valor medio di $\langle \Delta w_{lk} \rangle$ della regola di apprendimento per la componente l del neurone k troviamo

$$\langle \Delta w_{lk} \rangle = \frac{1}{m} \sum_{\nu(k)} (x_{l\nu} - w_{lk})$$

per cui, dal confronto di queste due espressioni, possiamo dedurre che la variazione media di un peso risulta

$$\langle \Delta w_{lk} \rangle \propto - \frac{\partial E}{\partial w_{lk}}$$

per cui durante l'apprendimento di fatto si minimizza la funzione $E(W)$ che è la somma delle distanze quadrate fra i pesi dei neuroni vincitori e gli

input che li rendono tali. Questa tecnica corrisponde al classico algoritmo di “k-means clustering”²⁰.

Alcuni problemi che si incontrano usando una rete di questo tipo:

- La funzione $E(W)$ è piena di minimi relativi: è facile capirlo osservando che diverse configurazioni dei neuroni possono risolvere il problema allo stesso modo. Perdipiù si possono anche scegliere gli esempi in modo malizioso tale da far sì che, se vengono dati alla rete sempre nello stesso ordine, il clustering cambi dopo ogni apprendimento. Inoltre abbiamo visto che la funzione è solamente continua a tratti e varia durante l'apprendimento: insomma non si presenta come una funzione “docile”;
- problema dei neuroni “morti”: un neurone che non vince mai non verrà mai aggiornato e sarà da considerare morto a tutti gli effetti;
- ci vogliono n neuroni per riconoscere n categorie mentre n neuroni potrebbero produrre 2^n output diversi. In sostanza questa rete tende ad implementare una “grandmother cell representation” in cui ogni neurone riconosce un particolare tipo di input;
- a questo è collegato il difetto dell'estrema “fragilità” di questa rete: se manca un neurone si perde tutta una categoria di esempi e questo nonostante ci siano molti neuroni (come detto sopra per n categorie ne potrebbero bastare $\log_2 n$).

9.5 Le reti di Kohonen

Nella rete che abbiamo esaminato punti vicini nello spazio degli input avevano l'immagine nello stesso neurone (a parte quelli dai due lati del confine di una cella di Voronoi) ma celle di Voronoi confinanti, e perciò vicine, non necessariamente avevano l'immagine in neuroni vicini, anche perché non è stato finora definito il concetto di “neuroni vicini”.

Introducendo una distanza fra i neuroni potremmo estendere il concetto di vicinanza nello spazio degli input da una sola cella di Voronoi a tutto lo spazio di input dato che ora anche input afferenti a celle di Voronoi adiacenti andrebbero a finire in neuroni adiacenti. In altre parole il concetto di distanza spaziale sarebbe in qualche modo riportato da tutto lo spazio degli input allo spazio delle sue rappresentazioni: lo spazio dei neuroni. Questo concetto di rappresentazione che conserva le distanze è ricorrente nella teoria delle reti neurali dove si parla di queste reti come di “topology preserving maps”.

²⁰nel 2004 Ding e He hanno dimostrato che il sottospazio dei centroidi di k clusters corrisponde ai primi $k-1$ termini dell'espansione della matrice di covarianza mostrando che c'è una intima relazione fra l'apprendimento non supervisionato e l'analisi delle componenti principali che possono essere usate per facilitare l'apprendimento.

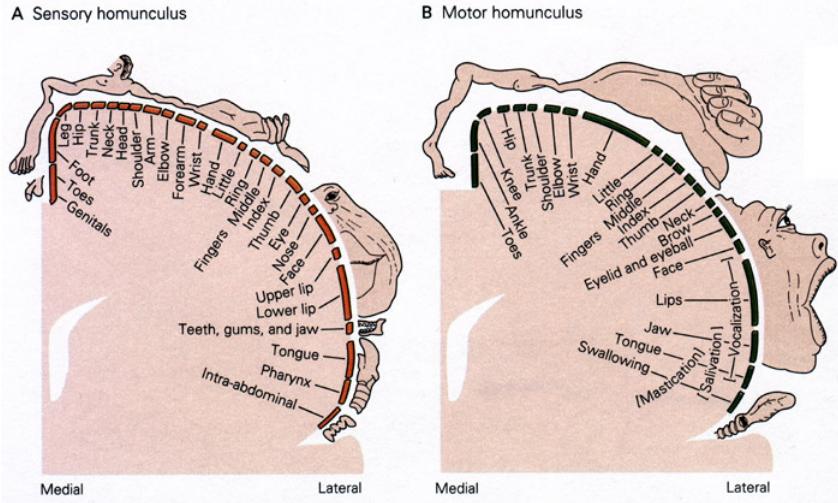


Figura 32: Homunculus: in questa figura sono rappresentate le connessioni fra le diverse parti del corpo e le relative aree negli emisferi cerebrali. A sinistra c'è l'homunculus sensoriale: più grande è disegnata la parte del cervello dedicata alla sua sensibilità; a destra c'è l'homunculus motorio: qui le aree sono proporzionali all'area del cervello dedicata al controllo motorio.

Questo concetto affonda le sue radici nelle reti biologiche dato che si sa, per esempio, che stimoli simili alla retina tendono a finire in aree vicine della corteccia cerebrale (mappa retinotopica fra retina e corteccia visiva). Altri esempi classici sono quelli della mappa sensoriale (sensotopica), vedi figura 32, e della mappa della corteccia uditiva (tonotopica).

Le reti di cui parleremo ora traggono origine proprio dalla fisiologia e da quando si cercava di capire come un algoritmo di apprendimento avrebbe potuto costruire una mappa di questo tipo nel cervello. Ai lavori più orientati a spiegare le mappe retinotopiche e tonotopica di Von Der Malsburg e Willshaw negli anni del 1970, che usavano la classica risposta a forma di "cappello messicano" (mexican hat, vedi figura 65), ha fatto seguito un lavoro più orientato verso le reti simulate fatto da Kohonen a partire dal 1982. Noi ci concentreremo su quest'ultimo approccio.

In sintesi l'ingrediente che dobbiamo aggiungere alle reti WTA del paragrafo precedente è il concetto di distanza fra i neuroni del secondo strato. Per definire la distanza dobbiamo prima decidere quante dimensioni dovrà avere lo spazio dei neuroni: per esempio potremo immaginare i neuroni disposti lungo una linea, perciò in uno spazio unidimensionale o formanti un reticolato bidimensionale o in spazi di dimensione più elevata.

In generale penseremo i neuroni disposti su di un reticolato discreto che potrà essere a una (\vec{w}_i), due (\vec{w}_{ij}) o più dimensioni ($\vec{w}_{ijk\dots k}$) per cui sarà

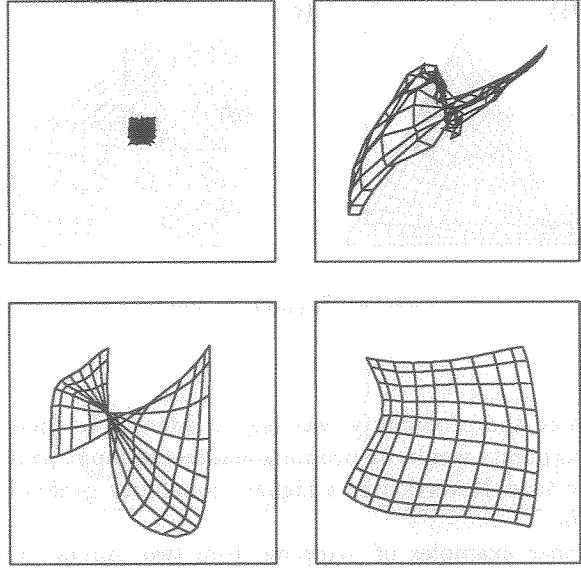


Figura 34: Evoluzione di una rete bidimensionale di Kohonen con esempi nel piano, anche qui dopo una fase di ordinamento avviene l'adattamento alla distribuzione degli esempi.

sempre facile definire i vicini di un neurone come quelli che si trovano ad un passo nel reticolo e così sarà automaticamente definita una distanza. La definizione non è rigorosa ma non ne ha nemmeno la pretesa.

L'impianto della rete sarà lo stesso di quello presentato nel paragrafo precedente, per cui una rete con apprendimento non supervisionato, con un'unica modifica per quanto riguarda la regola di apprendimento che ora è

$$\Delta \vec{w}_i = f(i, i^*)(\vec{x} - \vec{w}_i) \quad (23)$$

dove, al solito, i^* è il neurone che risponde maggiormente all'input e $f(i, i^*)$ è una funzione che regola l'apprendimento e che vale 1 quando $i = i^*$ e assume valori minori man mano che aumenta la distanza fra i e i^* . Una scelta tipica per una serie di neuroni a una o due dimensioni potrebbe essere

$$f(i, i^*) = e^{-\frac{d_{ii^*}^2}{2\sigma^2}} \quad d_{ii^*}^2 = \begin{cases} (i - i^*)^2 & \text{per } \vec{w}_i \\ (i - i^*)^2 + (j - j^*)^2 & \text{per } \vec{w}_{ij} \end{cases}$$

con σ che regola la larghezza dell'intervallo di neuroni che partecipano all'apprendimento; in figura 33 si vede l'apprendimento per una rete unidimensionale di neuroni. Chiaramente se $f(i, i^*) = \delta_{ii^*}$ si riottiene la rete WTA il che mostra come la rete di Kohonen sia una generalizzazione di questa. Dal punto di vista dell'apprendimento ora abbiamo che non solo il neurone vincente si avvicina allo stimolo ma anche i suoi vicini, sebbene in quantità minore. Possiamo intuitivamente pensare alla rete di neuroni come

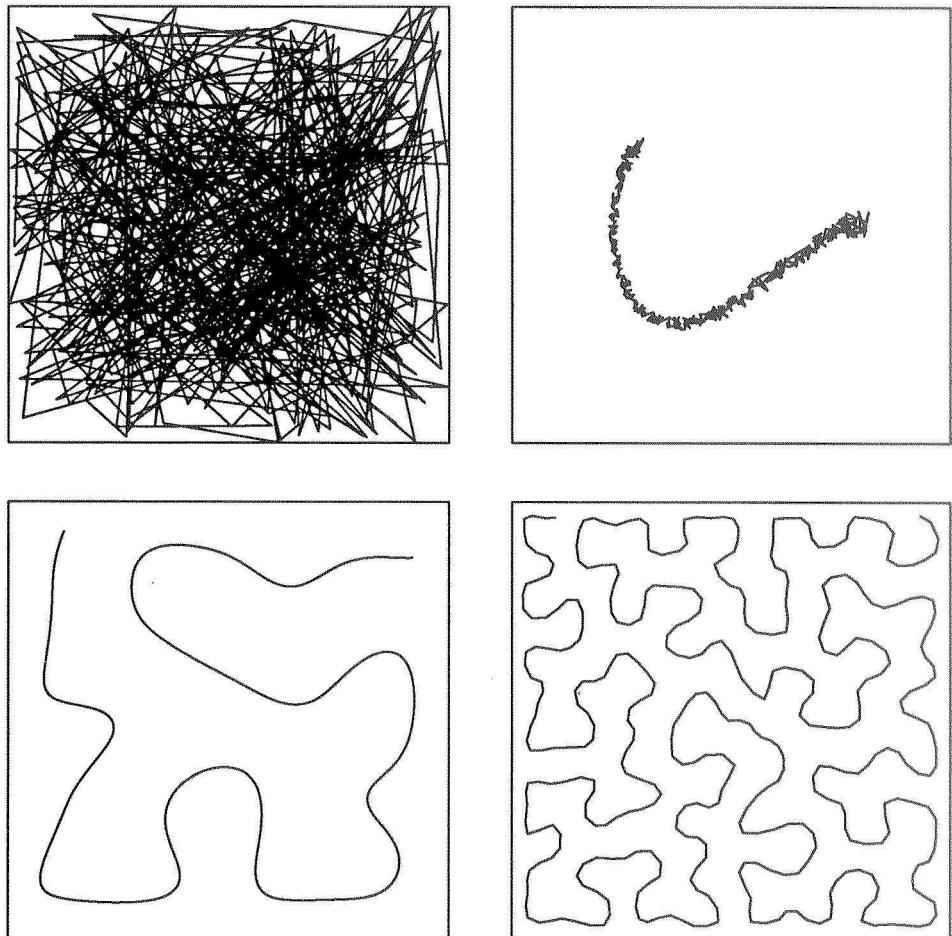


Figura 35: Evoluzione di una rete unidimensionale di neuroni con l'algoritmo di Kohonen, gli esempi sono distribuiti uniformemente nel quadrato. È mostrato il reticolo dei neuroni in 4 fasi successive dell'apprendimento; si noti come dapprima avviene l'ordinamento della rete e poi l'adattamento agli esempi. Nell'ultima fase, ad apprendimento terminato, i neuroni realizzano una rappresentazione unidimensionale di uno spazio bidimensionale: nella maggioranza dei casi ad input vicini corrisponderanno neuroni attivi vicini.

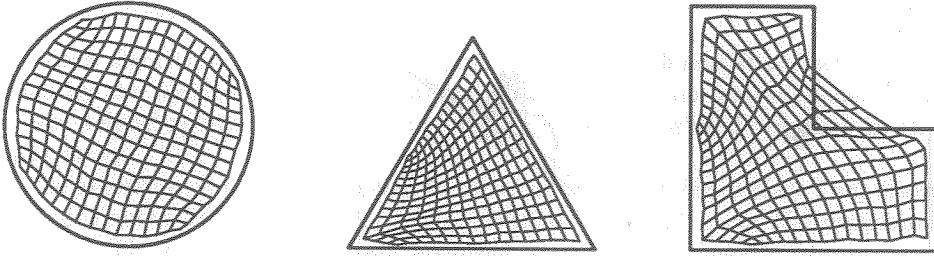


Figura 36: Evoluzione di una rete bidimensionale di Kohonen con esempi nel piano, gli input sono distribuiti uniformemente nelle 3 figure geometriche ed è indicato il reticolo dei neuroni alla fine dell'apprendimento; si noti come si adattano alla distribuzione degli input.

una rete elastica per cui far avvicinare il neurone vincente fa avvicinare, mediante l'elastico, anche i suoi vicini. Un vantaggio immediato è quello di eliminare il problema dei neuroni morti ma vedremo che non è l'unico.

Nelle figure 35, 34 e 36 si vedono alcuni esempi di apprendimento per reti di Kohonen ad una e due dimensioni. Si rammenta che date le dimensioni possiamo rappresentare nello stesso spazio sia gli esempi che i pesi dei neuroni. Dalle figure si vede che le promesse sono mantenute: input vicini vanno a finire, dopo l'apprendimento, in neuroni vicini. Naturalmente se lo spazio dei neuroni ha dimensione minore di quella degli input non tutte le distanze potranno essere rispettate ma vedia-

mo che la rete cerca di minimizzare i punti in cui ciò avviene. Si vede anche che la rete tende a disporsi in maniera che la densità dei neuroni riproduca, almeno parzialmente, la distribuzione degli input.

Oltre la regola di apprendimento (23) per utilizzare l'algoritmo di Kohonen è utile diminuire, durante l'apprendimento, sia ε che σ in maniera che all'inizio la rete sia influenzata molto dall'apprendimento, e si adatti alle caratteristiche più grossolane dei dati, e, alla fine invece, faccia solo dei raffinamenti locali.

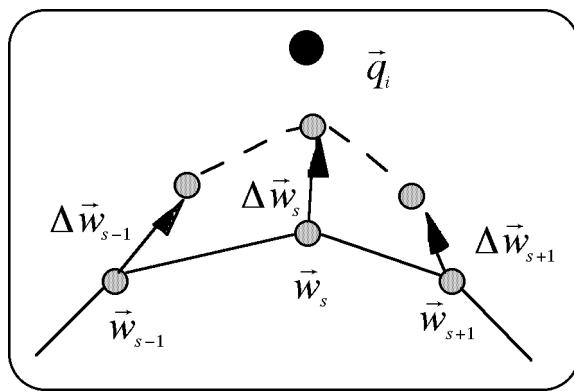


Figura 33: Apprendimento per una rete unidimensionale di neuroni: \vec{q}_i indica l'esempio, il neurone vincitore è \vec{w}_s e vengono modificati anche i pesi dei suoi primi vicini; dopo l'apprendimento i 3 neuroni si troveranno lungo la linea tratteggiata.

A questo punto diamo la traccia per scrivere un programma che esegue l'algoritmo di Kohonen. Per prima cosa occorre definire le dimensioni dello spazio dei neuroni (numero di indici) e di seguito la distanza fra due di essi; fissata in questo modo l'architettura della rete l'algoritmo è:

1. si inizializzano i pesi a valori casuali (tipicamente piccoli numeri non nulli) e i valori iniziali di ε e σ (per esempio $\varepsilon = 0.2$ e $\sigma = 18$);
2. si genera un numero casuale $1 \leq \nu \leq m$ per scegliere l'esempio da dare alla rete;
3. si dichiara neurone vincitore quello il cui peso è a distanza minore dall'input selezionato \vec{x}_ν (meglio usare la distanza fra neuroni e input anziché il prodotto scalare perché permette di evitare problemi di normalizzazione di input e pesi e, abbiamo visto, che nel caso di vettori normalizzati le cose sono equivalenti);
4. si modificano i pesi secondo la (23);
5. regolarmente, dopo un certo numero di iterazioni, si decrescono le costanti ε e σ , per esempio esponenzialmente (per esempio $\varepsilon' = 0.999\varepsilon$ e $\sigma' = 0.96\sigma$);
6. si verifica un criterio di stop (per esempio $\varepsilon \simeq 0$) e, se non raggiunto, si ritorna al punto 2.

Analizzare analiticamente questa rete è difficile; secondo una tecnica già collaudata proviamo a calcolare la variazione media di un vettore di pesi $\langle \Delta \vec{w}_i \rangle$ come la somma dei contributi (23) su tutti gli input possibili:

$$\langle \Delta \vec{w}_i \rangle = \frac{1}{m} \sum_{\nu=1}^m f(i, i^*(\nu))(\vec{x}_\nu - \vec{w}_i) \quad (24)$$

all'eventuale equilibrio avremo che per tutti i neuroni vale $\langle \Delta \vec{w}_i \rangle = 0$ ma, contrariamente a quanto visto per le reti lineari e la regola di Oja del paragrafo 9.2, questo sistema di equazioni risulta ora quasi sempre non risolubile. Le ragioni di questa difficoltà sono: la dipendenza non lineare di $i^*(\nu)$ da ν e la presenza stessa di $f(i, i^*(\nu))$.

L'unico caso in cui queste equazioni si riescono a risolvere è quello di una rete unidimensionale con un solo neurone di input. Anche per questo semplice caso tuttavia per risolvere l'equazione (ora in \mathbb{R})

$$\sum_{\nu=1}^m f(i, i^*(\nu))(x_\nu - w_i) = 0$$

occorre passare ad una approssimazione continua del sistema introducendo una distribuzione degli input $P(x)$ e trasformando la sommatoria in un integrale. Anche i neuroni diventano una distribuzione continua di densità $\delta(w)$

e si trova che si verifica la condizione di equilibrio quando

$$\delta(w) \propto P(x)^{\frac{2}{3}}$$

dunque anche in questo semplice caso la rappresentazione dei neuroni non è ideale e tende a sottostimare gli input più frequenti ($P(x) > 1$) e a sovrastimare quelli rari ($P(x) < 1$).

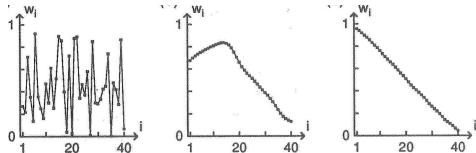


Figura 37: Apprendimento per una rete unidimensionale di 40 neuroni per input anch'essi unidimensionali.

Infine possiamo osservare dalla (24) che durante l'apprendimento queste reti in media discendono il gradiente della funzione

$$E(W) = \frac{1}{2m} \sum_{\nu=1}^m \sum_{i=1}^n f(i, i^*(\nu)) (\vec{x}_\nu - \vec{w}_i)^2$$

dato che facilmente si trova che

$$\langle \Delta \vec{w}_i \rangle = - \frac{\partial E(W)}{\partial \vec{w}_i}$$

Per cercare di capire, almeno fenomenologicamente, le caratteristiche di queste reti analizziamo cosa succede per una rete con h neuroni disposti in un reticolo unidimensionale come in figura 38. Possiamo definire

$$D = \sum_{i=2}^h |\vec{w}_i - \vec{w}_{i-1}| - |\vec{w}_h - \vec{w}_1| \geq 0$$

e facilmente si vede che $D = 0$ se, e solo se, i pesi sono: tutti allineati su una retta ed ordinati in ordine crescente.

Useremo D per indagare cosa succede durante l'apprendimento in questa rete con le ipotesi semplificatrici aggiuntive che gli input siano coppie di numeri reali (per permetterci di disegnare input e neuroni nel piano), che la funzione $f(i, i^*(\nu))$ sia una funzione a gradino che vale 1 per $i^*(\nu)$ e i suoi due primi vicini e 0 per tutti gli altri neuroni e infine che i pesi non siano tutti uguali (per permetterci di definire univocamente il neurone vincitore). Con queste ipotesi calcoliamo la variazione ΔD dopo un passo di apprendimento. Notiamo innanzitutto che, se per fare un esempio, all'input \vec{x} corrisponde $i^* = 3$, verranno modificati solo i pesi \vec{w}_2 , \vec{w}_3 e \vec{w}_4 per cui nella definizione di D vengono modificati solo i termini per $2 \leq i \leq 5$ e di conseguenza saranno gli unici che contribuiranno a ΔD , per cui, senza perdere generalità, possiamo limitarci a calcolare ΔD per una rete di 5 neuroni e con $i^* = 3$.

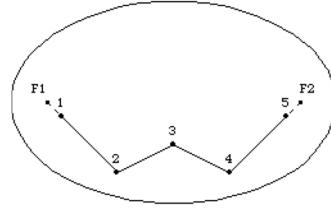


Figura 38: 5 neuroni di una rete unidimensionale con esempi nel piano.

Indicando con $\vec{w}'_i = \vec{w}_i + \varepsilon(\vec{x} - \vec{w}_i)$ i pesi modificati dall'apprendimento abbiamo

$$\Delta D = |\vec{w}'_2 - \vec{w}_1| + |\vec{w}'_3 - \vec{w}'_2| + |\vec{w}'_4 - \vec{w}'_3| + |\vec{w}'_5 - \vec{w}'_4| - \sum_{i=2}^5 |\vec{w}_i - \vec{w}_{i-1}|$$

facilmente si trova che $|\vec{w}'_i - \vec{w}'_{i-1}| = |(1-\varepsilon)(\vec{w}_i - \vec{w}_{i-1})| = (1-\varepsilon)|\vec{w}_i - \vec{w}_{i-1}|$ l'ultima uguaglianza derivando dal fatto che vale sempre $0 \leq \varepsilon \leq 1$. Dunque facilmente troviamo che

$$\Delta D = |\vec{w}'_2 - \vec{w}_1| + |\vec{w}_5 - \vec{w}'_4| - \varepsilon \left(\frac{|\vec{w}_2 - \vec{w}_1|}{\varepsilon} + |\vec{w}_3 - \vec{w}_2| + |\vec{w}_4 - \vec{w}_3| + \frac{|\vec{w}_5 - \vec{w}_4|}{\varepsilon} \right)$$

e infine

$$\begin{aligned} |\vec{w}'_2 - \vec{w}_1| &= \varepsilon |\vec{x} - \vec{w}_2 - \frac{\vec{w}_1 - \vec{w}_2}{\varepsilon}| \\ |\vec{w}_5 - \vec{w}'_4| &= \varepsilon |\vec{x} - \vec{w}_4 - \frac{\vec{w}_5 - \vec{w}_4}{\varepsilon}| \end{aligned}$$

per cui definendo le quantità indipendenti da \vec{x}

$$\begin{aligned} \vec{F}_1 &= \vec{w}_2 + \frac{\vec{w}_1 - \vec{w}_2}{\varepsilon} \\ \vec{F}_2 &= \vec{w}_4 + \frac{\vec{w}_5 - \vec{w}_4}{\varepsilon} \\ 2a &= \frac{|\vec{w}_2 - \vec{w}_1|}{\varepsilon} + |\vec{w}_3 - \vec{w}_2| + |\vec{w}_4 - \vec{w}_3| + \frac{|\vec{w}_5 - \vec{w}_4|}{\varepsilon} > 0 \end{aligned}$$

troviamo che la variazione di D in un passo di apprendimento è

$$\Delta D = \varepsilon (|\vec{x} - \vec{F}_1| + |\vec{x} - \vec{F}_2| - 2a)$$

Questa relazione mostra come varia D a seconda dell'input \vec{x} dato alla rete (ricordiamo però che \vec{x} necessariamente starà nella cella di Voronoi di \vec{w}_3 dato che questo è il neurone vincitore per l'ipotesi fatta all'inizio). In particolare per avere $\Delta D \leq 0$ occorre che \vec{x} soddisfi la relazione

$$|\vec{x} - \vec{F}_1| + |\vec{x} - \vec{F}_2| \leq 2a$$

cioè appartenga a un elisse di fuochi dati da \vec{F}_1 e \vec{F}_2 e semiasse maggiore a riprodotta in figura 38 (osserviamo che per ogni valore dei pesi $a > 0$ dato che abbiamo supposto che i pesi non siano tutti coincidenti). Lasciamo come esercizio dimostrare che tutti i pesi giacciono strettamente all'interno dell'elisse a meno che i pesi siano tutti allineati nel qual caso l'elisse diventa un segmento e i vari pesi giacciono sul confine cioè sul segmento stesso.

Esaminiamo ora il caso ancor più semplice di una rete con un input unidimensionale nel qual caso sia gli input x che i pesi w_i saranno dei semplici numeri reali. In questo caso l'elisse è diventata un segmento sull'asse reale. Supponiamo che i pesi siano ordinati, nel qual caso sappiamo che $D = 0$, allora possiamo dimostrare facilmente che $\Delta D = 0$. Infatti se w_3 è risultato essere il vincitore allora, dato che x appartiene alla sua cella di Voronoi,

necessariamente $w_2 < x < w_4$, e, dato che tutti i pesi sono compresi nel segmento che corrisponde all'elisse, a maggior ragione lo sarà x ; di conseguenza $\Delta D = 0$.

Questo risultato ci dimostra che una volta raggiunto lo stato ordinato la rete di neuroni vi resterà per ogni passo di apprendimento successivo. Come si dice nel linguaggio dei sistemi stocastici lo stato $D = 0$ è uno “stato assorbente”.

Un teorema molto generale ci garantisce che, se a ogni istante c’è una probabilità strettamente maggiore di 0 di arrivare allo stato assorbente, questo verrà raggiunto, dopo un numero sufficientemente grande di iterazioni, con probabilità 1. Ma la nostra condizione $a \geq 0$ mostra che ad ogni istante esiste un insieme finito di x che fa diminuire D e si può costruire una sequenza di questi x che porta allo stato assorbente; dunque ad ogni istante esiste una probabilità finita che soddisfa il teorema citato; di conseguenza la convergenza allo stato assorbente è garantita.

Riassumendo abbiamo dimostrato che, per questa rete, lo stato di neuroni ordinati è uno stato assorbente e che in ogni istante esiste una probabilità non nulla di dirigervisi, dunque che sarà raggiunto con probabilità 1. In altre parole la rete inizia con pesi casuali e $D > 0$ e dopo un certo periodo avrà messo i neuroni in ordine crescente e da quel momento in poi resteranno ordinati e l'apprendimento adatterà la rete alla distribuzione degli input. Almeno per questa semplice rete esistono dimostrabilmente due fasi ben distinte dell'apprendimento:

- una fase di ordinamento nella quale D decresce fino a raggiungere $D = 0$ con probabilità 1;
- una fase di adattamento quando si è raggiunto lo stato ordinato e la rete si adatta alla distribuzione degli input $P(x)$.

Nel caso generale di input a più dimensioni non è facile definire nemmeno il significato di rete ordinata, tantomeno dimostrare che il suo raggiungimento si ottiene sempre nè che sia assorbente. Al tempo stesso questo è esattamente quanto si osserva dal punto di vista fenomenologico quando si fanno simulazioni numeriche, ma al contrario del caso unidimensionale, non c’è nessuna garanzia di arrivare a una fase ordinata. Possiamo cercare di intuire come vadano le cose ritornando a esaminare le elissi per una rete con input a due dimensioni. All'inizio i pesi sono disordinati e le elissi sono quasi circolari per cui danno una forte probabilità che \vec{x} cada all'interno il che fa calare D e tende a portare la rete verso uno stato più ordinato. Supponiamo però che si sia raggiunto uno stato ordinato $D = 0$ a questo punto abbiamo visto che l'elissi è diventata un segmento e la probabilità che \vec{x} vi sia contenuto è 0 (un segmento ha misura nulla in uno spazio bidimensionale) per cui avremo che D tornerà ad aumentare senza che lo stato ordinato sia assorbente. Simulazioni numeriche confermano questo risultato intuitivo.

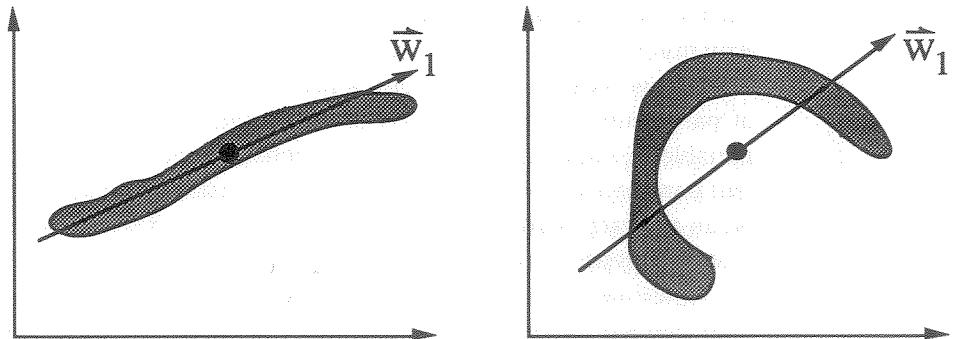


Figura 39: Esempi schematici di distribuzione di esempi, \vec{w}_1 indica la prima componente principale. Nel primo caso le cose vanno bene, nel secondo ci vorrebbero delle coordinate curvilinee.

Concludiamo l'argomento delle reti di Kohonen con una osservazione. È fuor di dubbio che la riduzione dimensionale sia una proprietà interessante e nell'analisi delle componenti principali (PCA, vedi appendice A.4) questa si raggiungeva tutto sommato facilmente. Nel caso però di dati non disposti in modo lineare, come nell'esempio in figura 39 o in quello patologico riportato della figura 89, è evidente che la PCA non porta nessuna riduzione dimensionale. Una trasformazione più complessa dello spazio di input potrebbe definire delle coordinate curvilinee che chiaramente potrebbero portare ad una riduzione dimensionale come si intuisce nell'esempio in figura 40. Si parla allora per queste trasformazioni non lineari di "principal curves"; più in generale se usassimo delle superfici al posto delle curve si parlerebbe di "principal manifolds".

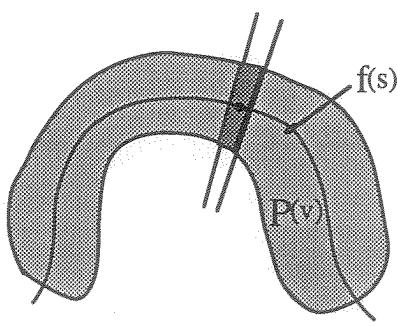


Figura 40: Rappresentazione schematica di una principal curve indicata con $f(s)$.

Le componenti principali si potevano ottenere trovando la direzione che massimizzava la varianza dei dati o, come già osservato nell'appendice A.4, minimizzando la varianza nel sottospazio ad essa ortogonale. Qui potremmo usare un concetto simile, minimizzare la varianza *locale*, per definire una principal curve o manifold. In altre parole si può definire la principal curve, come mostrato nell'esempio in figura 40, considerando dei trattini infinitesimi di coordinata curvilinea e definendo la posizione della

principal curve come quel punto che minimizza la "varianza locale" dei dati che corrispondono a quel valore della coordinata curvilinea. Minimizzando

la varianza locale la curva tende a passare per il “baricentro locale” dei dati e dunque a seguire i dati, in questo modo verrà massimizzata la “varianza globale” dei dati lungo tutta la coordinata curvilinea.

Mostriamo ora che la rete di Kohonen tende a minimizzare proprio una quantità del genere, sebbene in modo approssimato. Se guardiamo infatti la funzione minimizzata da Kohonen alla fine dell'apprendimento (quando vale $f(i, i^*(\nu)) = \delta_{ii^*(\nu)}$) troviamo che essa è diventata

$$E(W) = \frac{1}{2m} \sum_{\nu=1}^m (\vec{x}_\nu - \vec{w}_{i^*(\nu)})^2$$

per cui la somma delle distanze al quadrato fra un neurone e tutti gli input che selezionano quel neurone come vincitore. Per cui minimizziamo, *localmente* la somma delle distanze al quadrato, cioè la varianza. Possiamo così capire perché si usa pensare alle reti di Kohonen come ad approssimazioni discrete ai principal manifolds e la dimensione del reticolo della rete corrisponde alle dimensioni della superficie (o manifold) che si usa per approssimare i dati in uno spazio con pochissime dimensioni.

Prima di finire questo paragrafo due osservazioni di carattere generale. La prima riguarda il concetto di vicinanza spaziale fra esempi nello spazio degli input. Questa vicinanza non va confusa con quella che c'è, per esempio, fra i pixel adiacenti di una telecamera che, a priori, rappresentano dimensioni diverse dello spazio degli input. Nondimeno pixel vicini di una telecamera che riprenda immagini naturali sono in realtà estremamente correlati. Per convincersene basta guardare il mondo intorno a noi per rendersi conto che le immagini sono formate soprattutto da aree più o meno uniformi. Per esempio uno potrebbe stimare il contenuto di un pixel dalla media dei suoi adiacenti e, nella maggior parte dei casi, non sbaglierebbe di molto. Dunque in questo caso pixel adiacenti, pur essendo teoricamente dimensioni diverse, risultano quasi sempre uguali e per esempi di immagini naturali pixel vicini tenderanno ad avere lo stesso valore. Dunque anche la vicinanza fra pixel dovrà avere un ruolo nella vicinanza fra esempi.

Infine riguardo alle dimensioni dello spazio realmente occupato dagli input possiamo intuire quanto possa essere piccolo pensando a questo esempio. Supponiamo di generare una serie di immagini sintetiche muovendo in vari modi k oggetti nello spazio e proiettando la loro immagine su una telecamera; è chiaro che avremo un numero di gradi di libertà per generare le immagini che sarà proporzionale a k . Di conseguenza le dimensioni dello spazio che contiene le immagini sintetiche generate sarà qualcosa di più vicino a k che non al numero, grande a piacere, di pixel della telecamera.

Per finire osserviamo che in applicazioni pratiche una rete ibrida potrebbe rappresentare una soluzione molto interessante per un dato problema.

Prima si mette una rete di Kohonen che fa apprendimento non supervisionato ed esegue la riduzione dimensionale dei dati con il principio dei principal manifolds. Poi si aggiunge uno strato successivo il cui compito sarà di riconoscere l'input, per esempio con apprendimento supervisionato e back-propagation. Solo che ora lo si fa sullo spazio di input ridotto generato dalla rete di Kohonen. Questo approccio in pratica funziona abbastanza bene, risulta fondato su ragionevoli basi teoriche, e, alla prova dei fatti, risulta di *ordini di grandezza* più veloce di un apprendimento tradizionale con la back-propagation sullo spazio originale degli input, soprattutto perché ha ridotto sostanzialmente la dimensione dello spazio di input e di conseguenza il numero di pesi coinvolto.

10 Reti neurali e teoria dell'informazione

Nei sistemi biologici sono noti molti fenomeni di autorganizzazione dei neuroni come per esempio nel sistema visivo umano nel primo periodo di vita dopo la nascita (vedi nel paragrafo 13.1.3). Dal punto di vista di una rete neurale simulata l'autorganizzazione si associa all'apprendimento non supervisionato per una rete di neuroni di McCulloch e Pitts e sin dai primi lavori di Barlow nel 1960 sono stati proposti algoritmi di apprendimento basati sui concetti della teoria dell'informazione (appendice A.5). Dopo aver presentato un caso biologico esamineremo alcuni casi di applicazione della teoria dell'informazione alle reti neurali, il primo dedicato all'apprendimento del Perceptron continuo, il secondo ad uno strato di h Perceptron per concludere con un semplice esempio di algoritmo di apprendimento.

10.1 Un semplice esempio biologico: l'occhio della mosca

Le grandi cellule monopolari (LMC) dell'occhio composto della mosca sono neuroni dotati di risposta graduale che dipende dall'intensità luminosa x che li colpisce. Essi adattano il guadagno w e la soglia θ alla distribuzione dell'intensità luminosa dell'input $p(x)$. Supponiamo di rappresentarli con un neurone di McCulloch e Pitts che produce una risposta y data da

$$y = f(xw - \theta)$$

con una funzione di trasferimento continua e facciamo l'ipotesi che il neurone adatti i suoi parametri in maniera da massimizzare l'informazione trasferita. Questo si ottiene massimizzando l'entropia differenziale dell'output $s(y)$ (59) (appendice A.5) il che a sua volta si ottiene se la distribuzione di y è uniforme

$$p(y) = \frac{1}{a} .$$

Dato che la probabilità si conserva deve valere

$$p(x)dx = p(y)dy = \frac{1}{a}dy$$

dalla quale

$$\frac{dy}{dx} = ap(x)$$

che può essere facilmente integrata dando

$$y = f(wx - \theta) = a \int p(x)dx$$

che dice che l'informazione in y è massima se la funzione implementata dal neurone è uguale alla distribuzione cumulativa di $p(x)$. Non sempre è possibile realizzare questa condizione ma ci si può aspettare che, se si vuole

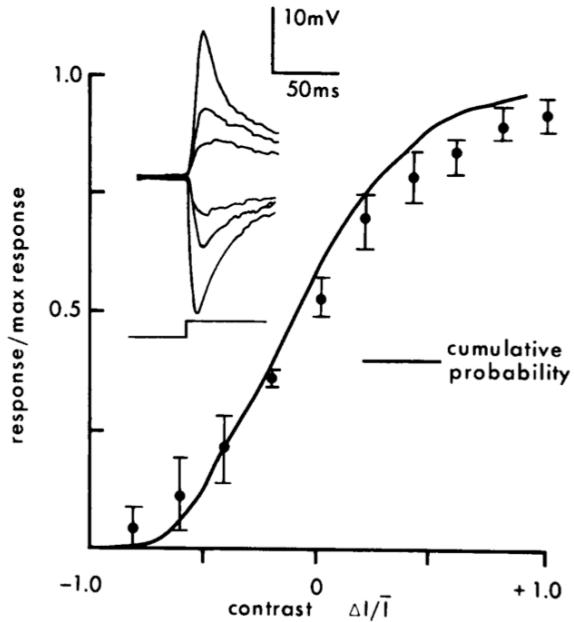


Figura 41: Curva di risposta di neuroni LMC dell’occhio di mosca

massimizzare l’informazione in y , i parametri w e θ saranno scelti in modo da avvicinarsi il più possibile a questa condizione.

Nel 1981 Laughlin [9] si è proposto di verificare se la risposta dei neuroni LMC realizza questa condizione: per far questo per prima cosa ha misurato sperimentalmente la relazione fra intensità luminosa x e differenza di potenziale prodotta $y = f(x)$. A questo punto il buon Laughlin è andato a fotografare le scene naturali dell’habitat della mosca (boschi, campagne etc.) dai quali ha ricavato una stima della distribuzione $p(x)$ osservata della quale ha poi potuto calcolare la relativa cumulativa. Ha poi confrontato questa curva con la curva misurata sperimentalmente di $y = f(x)$ e il risultato è riprodotto in figura 41 dove la curva continua è la cumulativa della distribuzione $p(x)$ e i punti con le barre d’errore sono le differenze di potenziale prodotte dalle LMC in funzione dell’input. Si osserva che c’è un’ottima corrispondenza il che fa pensare che i parametri dei neuroni LMC realizzino con buona approssimazione la condizione di massimizzare l’entropia differenziale dell’output $s(y)$ ²¹.

²¹una nota tecnica, Laughlin in realtà non ha lavorato con l’intensità luminosa x ma bensì con il contrasto c definito come

$$c = \frac{x}{\mu_x} - 1$$

dove μ_x è il valore medio di x . Si vede che il contrasto dipende linearmente da x ed è limitato inferiormente a -1 quando $x = 0$, e dunque tutto il discorso su c è una semplice variazione di quello su x che noi abbiamo fatto per semplicità di esposizione.

10.2 Un altro algoritmo di apprendimento per il Perceptron

Nel caso del Perceptron continuo, trattato nel paragrafo 5.2, la discesa lungo il gradiente della funzione d'errore $E(\vec{w})$ produce (11) un termine $f'(\vec{x}_\nu \cdot \vec{w})$ che non appare nella regola del caso discreto (7) e che in pratica ha l'effetto di rallentare moltissimo l'apprendimento. Ora vediamo come si può ovviare a questo problema usando un'altra definizione della funzione d'errore $E(\vec{w})$. A priori non c'è nessuna ragione per usare una funzione d'errore quadratica come quella usata tradizionalmente: qualsiasi funzione che raggiunga il minimo quando $\xi_\nu = y_\nu$ per tutti i ν può andare bene.

Una funzione con caratteristiche molto interessanti si trova pensando alle ξ_ν e alle y_ν come a delle variabili stocastiche (casuali) e definendo come funzione d'errore per il Perceptron la distanza di Kullback Leibler fra le due distribuzioni. Penseremo dunque ai valori $\xi_1, \xi_2, \dots, \xi_m$ come a uno dei possibili 2^m valori che può assumere un vettore $\vec{z} = (z_1, z_2, \dots, z_m)$ e la distribuzione delle ξ_ν , $P_\xi(\vec{z})$, è una distribuzione che vale 1 quando $\vec{z} = (\xi_1, \xi_2, \dots, \xi_m)$ e 0 per tutti gli altri \vec{z} ; in modo del tutto analogo definiamo la distribuzione delle y_ν , $P_y(\vec{z})$. La distanza fra le due distribuzioni vale

$$D_{KL}(P_\xi(\vec{z}), P_y(\vec{z})) = \sum_{\vec{z}=(-1,-1,\dots,-1)}^{(1,1,\dots,1)} P_\xi(\vec{z}) \log_2 \frac{P_\xi(\vec{z})}{P_y(\vec{z})}$$

dove la somma è estesa a tutti i possibili 2^m valori di \vec{z} . Vista la definizione è chiaro che, essendo tutte le ξ_ν indipendenti fra loro, la distribuzione $P_\xi(\vec{z})$ è fattorizzata

$$P_\xi(\vec{z}) = p_\xi(z_1)p_\xi(z_2)\cdots p_\xi(z_m)$$

ed una relazione identica vale per la $P_y(\vec{z})$. Con semplice algebra si trova

$$D_{KL}(P_\xi(\vec{z}), P_y(\vec{z})) = \sum_{\nu=1}^m \sum_{z_\nu=-1}^1 p_\xi(z_\nu) \log_2 \frac{p_\xi(z_\nu)}{p_y(z_\nu)}$$

In pratica la somma delle distanze di Kullback Leibler per i singoli input. Ogni variabile assume i valori 1 e -1 con probabilità date da

$$\begin{array}{lll} & \xi_\nu = 1 & \xi_\nu = -1 \\ p_\xi(z_\nu = 1) & = & \frac{1+\xi_\nu}{2} & 1 & 0 \\ p_\xi(z_\nu = -1) & = & \frac{1-\xi_\nu}{2} & 0 & 1 \end{array}$$

e analogamente per y_ν ; ovviamente $p_\xi(z_\nu = 1) + p_\xi(z_\nu = -1) = 1$. Con queste definizioni possiamo finalmente scrivere la distanza di Kullback Leibler in funzione delle ξ_ν e y_ν

$$D_{KL}(P_\xi(\vec{z}), P_y(\vec{z})) = \frac{1}{2} \sum_{\nu=1}^m (1 - \xi_\nu) \log_2 \frac{1 - \xi_\nu}{1 - y_\nu} + (1 + \xi_\nu) \log_2 \frac{1 + \xi_\nu}{1 + y_\nu}$$

che interpretiamo come la nuova funzione d'errore per il Perceptron. La dipendenza dei pesi si ha attraverso i valori di $y_\nu = f(\vec{x}_\nu \cdot \vec{w})$. Dalle proprietà della distanza di Kullback Leibler discende che $D_{KL}(P_\xi(\vec{z}), P_y(\vec{z})) \geq 0$ e l'uguaglianza vale solamente se $P_\xi(\vec{z}) = P_y(\vec{z})$ cioè se $\xi_\nu = y_\nu$ per tutti i ν ; inoltre ha la piacevole proprietà di divergere se uno degli output y_ν ha il valore opposto a quello desiderato (al contrario di quella quadratica che per ogni output errato aumenta solo di 2). Se per funzione di trasferimento prendiamo la tangente iperbolica, con un semplice esercizio di derivazione (fatelo!) si trova che

$$-\frac{\partial D_{KL}(P_\xi(\vec{z}), P_y(\vec{z}))}{\partial w_i} = \log_2 e \sum_{\nu=1}^m (\xi_\nu - y_\nu) x_{i\nu}$$

e per questa scelta della funzione d'errore gli algoritmi del Perceptron discreto e continuo coincidono essendo scomparso il fastidioso fattore $f'(\vec{x}_\nu \cdot \vec{w})$ che spesso rallenta l'apprendimento quando un esempio è classificato nel modo sbagliato. Con questa forma della funzione d'errore si possono risolvere problemi che risultano irresolubili per la funzione d'errore quadratica.

10.3 Uno strato di h Perceptron

Applichiamo ora le idee più generali della teoria dell'informazione al caso, particolarmente semplice, di uno strato di Perceptron binari (per una trattazione più dettagliata si veda nell'articolo di Nadal e Parga [11]).

Ispirandosi ai sistemi biologici e cercando di riprodurne e analizzarne il funzionamento sono stati proposti diversi modelli di apprendimento, prima di descriverli iniziamo con alcune definizioni generali.

Supponiamo di avere un sistema in cui vengono prodotti degli input continui $\vec{x} \in \mathbb{R}^n$ che vengono passati a una rete che produce degli output discreti $\vec{y} \in \{\pm 1\}^h$. Gli input hanno una distribuzione $p(\vec{x})$ mentre gli output sono distribuiti con $p(\vec{y})$ che è la distribuzione marginale di $p(\vec{x}, \vec{y})$

$$p(\vec{y}) = \sum_{\vec{x}} p(\vec{x}, \vec{y})$$

ed inoltre

$$p(\vec{x}, \vec{y}) = p(\vec{y}|\vec{x})p(\vec{x})$$

dove $p(\vec{y}|\vec{x})$ è la probabilità condizionale di osservare \vec{y} quando l'input alla rete è stato \vec{x} , in essa sono racchiuse le caratteristiche della rete come architettura, numero di neuroni, pesi etc.. In appendice A.5 abbiamo definito la mutua informazione

$$\begin{aligned} M(\vec{x}, \vec{y}) &= \sum_{\vec{x}} \sum_{\vec{y}} p(\vec{x}, \vec{y}) \log_2 \frac{p(\vec{x}, \vec{y})}{p(\vec{x})p(\vec{y})} = \\ &= \sum_{\vec{x}} \sum_{\vec{y}} p(\vec{y}|\vec{x})p(\vec{x}) \log_2 \frac{p(\vec{y}|\vec{x})}{p(\vec{y})} \end{aligned}$$

dove abbiamo applicato la definizione di probabilità condizionale. $M(\vec{x}, \vec{y})$ è limitata superiormente dall'entropia della sorgente²² (per convincersene basta ricordare la figura 93 che rappresenta la mutua informazione come intersezione delle due entropie)

$$M(\vec{x}, \vec{y}) \leq s(x) = \int p(x) \log_2 \frac{1}{p(x)} dx .$$

Queste definizioni sono del tutto generali e valgono per ogni tipo di rete, ora appliciamole al nostro caso particolare: uno strato di neuroni composto da h Perceptron binari che producono l'output in base alla regola deterministica $y_i = \text{sgn}(\vec{x} \cdot \vec{w}_i)$. Per codificare questa regola deterministica in linguaggio probabilistico possiamo dire che la probabilità che y_i e $\vec{x} \cdot \vec{w}_i$ abbiano lo stesso segno è 1, mentre quella che abbiano segno opposto è 0 cioè $p(y_i|\vec{x}) = \Theta(y_i \vec{x} \cdot \vec{w}_i)$ dove la funzione Θ è quella del capitolo 3. In questo caso la distribuzione degli h output è univocamente determinata dagli input e la possiamo scrivere

$$p(\vec{y}|\vec{x}) = \prod_{i=1}^h \Theta(y_i \vec{x} \cdot \vec{w}_i) \in \{0, 1\}$$

che, fa sì che, per un dato input \vec{x} , solo l'output corrispondente \vec{y} abbia probabilità 1 mentre tutti gli altri $2^h - 1$ output possibili hanno probabilità nulla. Possiamo sostituire questa definizione della probabilità condizionale $p(\vec{y}|\vec{x})$ nella definizione di $M(\vec{x}, \vec{y})$ che, invertendo l'ordine delle somme, si scrive

$$M(\vec{x}, \vec{y}) = \sum_{\vec{y}} \sum_{\vec{x}} p(\vec{y}|\vec{x}) p(\vec{x}) \log_2 \frac{p(\vec{y}|\vec{x})}{p(\vec{y})}$$

dove possiamo osservare che dato che $p(\vec{y}|\vec{x}) \in \{0, 1\}$ nei termini diversi da 0 nella somma su \vec{x} il termine con il logaritmo ha sempre lo stesso valore $\log_2 \frac{1}{p(\vec{y})}$ e lo possiamo mettere fuori dalla somma su \vec{x} la quale diventa

$$\sum_{\vec{x}} p(\vec{y}|\vec{x}) p(\vec{x}) = \sum_{\vec{x}} p(\vec{x}, \vec{y}) = p(\vec{y})$$

la probabilità marginale, in conclusione la mutua informazione diventa

$$M(\vec{x}, \vec{y}) = \sum_{\vec{y}} p(\vec{y}) \log_2 \frac{1}{p(\vec{y})} = S(\vec{y})$$

e dunque, come d'altra parte era logico aspettarsi, per una rete deterministica la mutua informazione è uguale all'entropia dell'output. Possiamo applicare il teorema dimostrato nell'appendice A.5 che da il limite superiore

²²Per input continui l'entropia differenziale della sorgente potrebbe risultare infinita (vedi appendice A.5) se non si introduce una risoluzione minima per \vec{x} .

all'entropia $S(\vec{y}) \leq \log_2 W$. Chiameremo questo limite superiore la *capacità informativa* della rete \mathcal{C} che possiamo anche definire in maniera più generale come

$$\mathcal{C} = \max_{p(\vec{x}), \vec{w}_1, \dots, \vec{w}_n} M(\vec{x}, \vec{y}) \quad (25)$$

e rappresenta la massima quantità di informazione che la rete può trasmettere. Chiaramente dipende dalla rete o, più precisamente, dalla probabilità condizionale $p(\vec{y}|\vec{x})$. In questo caso essendo l'output binario sappiamo che potrà avere al più $W \leq 2^h$ output diversi e concludiamo

$$M(\vec{x}, \vec{y}) = S(\vec{y}) \leq \mathcal{C} = \log_2 W \leq \log_2 2^h = h$$

Questo limite superiore può essere fatto più stringente osservando che in generale non tutti i 2^h output sono stati possibili per la nostra rete. Per convincersene ricordiamo che nel paragrafo 7.1 avevamo definito la capacità di un Perceptron contando il numero di funzioni diverse che esso poteva implementare per m esempi in posizione generale in \mathbb{R}^n che vale (14)

$$C(m, n) = 2 \sum_{k=0}^{n-1} \binom{m-1}{k} \leq 2^m$$

e ad ognuna di queste funzioni corrisponde una serie di output $(1, -1, \dots, 1, 1) \in \{\pm 1\}^m$ forniti dal Perceptron per tutti gli m esempi e identifica univocamente la funzione realizzata dal vettore dei pesi \vec{w} ; in generale non tutte le 2^m configurazioni sono realizzabili dal Perceptron.

Per applicare questo risultato al nostro caso osserviamo che, per la simmetria del prodotto scalare $\vec{x}_\nu \cdot \vec{w}$, posso anche pensare ai vari output $y_\nu = \text{sgn}(\vec{x}_\nu \cdot \vec{w})$ come ad un insieme di m Perceptron, di pesi $\vec{x}_1, \dots, \vec{x}_m$ che guardano in parallelo all'input \vec{w} ! Ad ogni funzione diversa del Perceptron 'singolo' corrispondono biunivocamente le funzioni degli m Perceptron del secondo caso. Se ne deduce che uno strato di h Perceptron possono realizzare al più $C(h, n)$ funzioni con configurazioni diverse, dunque $W = C(h, n)$ e possiamo scrivere il limite superiore all'entropia di \vec{y}

$$S(\vec{y}) \leq \mathcal{C} = \log_2 C(h, n) = \begin{cases} h & \text{per } h \leq n \\ < h & \text{per } h > n \end{cases}$$

Dato che non è facile calcolare $\log_2 C(h, n)$ si può aggirare l'ostacolo calcolando questa quantità nel limite $n, h \rightarrow \infty$ e in questo caso invece della capacità \mathcal{C} conviene calcolare la capacità informativa *per input* $\frac{c}{n}$ (anche perché l'informazione in ingresso cresce al crescere di n) in funzione del rapporto (costante) fra numero di neuroni e input $\alpha = \frac{h}{n}$. Si trova che

$$\lim_{n \rightarrow \infty} \frac{c}{n} := c = \begin{cases} \alpha & \text{per } \alpha \leq 2 \\ \log_2 \alpha + (1 - \alpha) \log_2 (1 - \frac{1}{\alpha}) & \text{per } \alpha \geq 2 \end{cases}$$

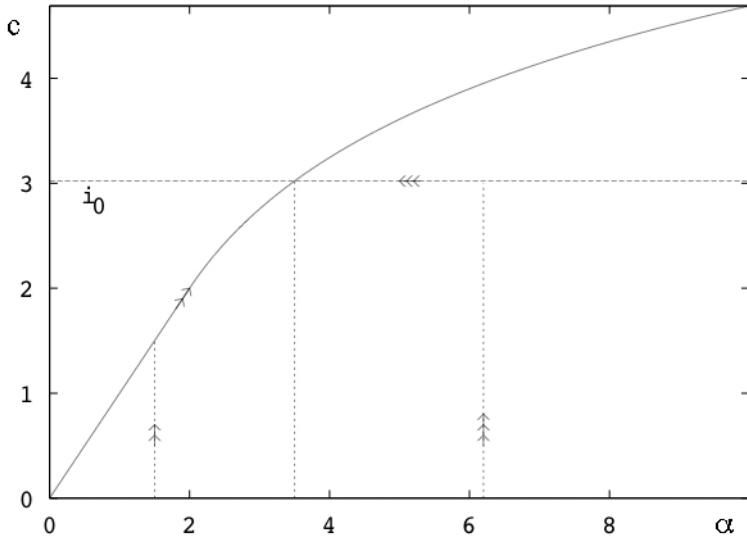


Figura 42: Capacità per bit c per uno strato di Perceptron binari in funzione del rapporto $\alpha = \frac{h}{n}$ fra neuroni e numero di input. c rappresenta il limite superiore di $\frac{S(\vec{y})}{n}$ per cui per un dato α la curva da il range possibile per $\frac{S(\vec{y})}{n}$

il cui andamento è riprodotto in figura 42.

Questo risultato ci dice che, per questo tipo di reti, la capacità informativa per bit di input c è indipendente dal valore dei pesi (dato che C (25) è il massimo per tutti i pesi \vec{w}) e dipende solo da α e cioè dalle dimensioni della rete n e h .

Dall'andamento di c in figura 42 possiamo capire meglio le relazioni fra informazione in ingresso e architettura della rete: supponiamo che gli input abbiano un entropia di 3 bit di informazione per ogni input (ricordiamo che questo è possibile dato che gli input sono continui); questo valore è marcato dalla linea tratteggiata orizzontale. Esaminiamo 3 possibili architetture corrispondenti a tre valori di α indicati da altrettante linee tratteggiate verticali. Nel primo caso la nostra rete ha $\alpha < 2$ e comunque si scelgano i pesi la rete non potrà trasmettere più del corrispondente valore di c , circa 1.5 bit per input. Nel secondo caso se la nostra rete ha $\alpha = 3.4$, linea verticale centrale, abbiamo una rete che può trasmettere in modo ottimale tutta l'informazione che arriva, purché naturalmente i pesi siano stati scelti opportunamente. Infine nell'ultimo caso $\alpha > 6$ anche se in teoria la rete potrebbe trasmettere più di 3 bit per input sarà comunque limitata dall'informazione che proviene dall'input. È facile visualizzare mentalmente questi casi pensando alla rappresentazione grafica della mutua informazione di figura 93.

Da queste osservazioni si vede come ci siano due elementi concorrenti:

da un lato l'architettura, fissando α e il conseguente valore di c stabilisce la massima quantità di informazione che questo tipo di rete può trasmettere. Dall'altro, l'apprendimento, permette di raggiungere (nel caso ottimale) il valore massimo c della mutua informazione.

Questo ci fornisce un interessante prospettiva biologica: dato che è ragionevole pensare che l'architettura di una rete sia fissa questo ci induce a pensare che l'evoluzione agisca sull'architettura, ottimizzando il valore di α all'informazione che il sistema può trovare in ingresso. Dall'altro lato l'apprendimento, adattando i pesi della rete, permette di raggiungere il massimo valore della mutua informazione fra input e output raggiungendo il limite superiore c .

10.4 Algoritmi di apprendimento basati sulla teoria dell'informazione

Iniziamo con l'esame di alcuni principi guida, basati sulla teoria dell'informazione, che si usano per trovare algoritmi di apprendimento non supervisionato per le reti neurali:

- *Infomax*: questo principio richiede di massimizzare la mutua informazione fra input e output $M(\vec{x}, \vec{y})$. Osserviamo che la mutua informazione dipende sia dai pesi che dalla distribuzione degli input $p(\vec{x})$; ne vedremo un esempio fra poco.
- *Minimizzazione della ridondanza*: in questo caso si vuole minimizzare la ridondanza definita come

$$R = \mathcal{C} - M(\vec{x}, \vec{y}) \geq 0$$

che vale 0 quando la rete trasmette la massima informazione $M(\vec{x}, \vec{y}) = \mathcal{C}$. Dato che abbiamo visto che per uno strato di Perceptron la capacità \mathcal{C} non dipende dai pesi ma solo dall'architettura, minimizzare la ridondanza è equivalente a massimizzare la mutua informazione, dunque questo principio porta allo stesso algoritmo dell'Infomax. Notiamo che per reti diverse da quella considerata in generale questi due algoritmi non sono equivalenti. Osserviamo infine che se fosse permesso di modificare l'architettura potremmo ridurre la ridondanza riducendo il valore di α (e di conseguenza di c) e adattando la rete all'ambiente, cioè all'informazione presente negli input.

- *Fattorizzazione della $p(\vec{y})$* : in questo caso si richiede che la distribuzione di output sia più simile possibile alla distribuzione fattorizzata dei singoli Perceptron dato che così si massimizza l'entropia dell'output.

In altre parole si chiede di minimizzare la distanza di Kullback Leibler

$$\begin{aligned} D_{KL}(p(\vec{y}), \prod_{i=1}^h p(y_i)) &= \sum_{\vec{y}} p(\vec{y}) \log_2 \frac{p(\vec{y})}{\prod_{i=1}^h p(y_i)} \\ &= \sum_{\vec{y}} p(\vec{y}) \log_2 p(\vec{y}) + \sum_{i=1}^h \sum_{\vec{y}} p(\vec{y}) \log_2 \frac{1}{p(y_i)} \end{aligned}$$

dove il primo termine è chiaramente $-S(\vec{y})$. Osservando che la probabilità marginale di una componente dell'output si può scrivere $p(y_i) = \sum_{y_1} \cdots \sum_{y_{i-1}} \sum_{y_{i+1}} \cdots \sum_{y_h} p(\vec{y})$, troviamo per il secondo termine

$$\sum_{i=1}^h \sum_{\vec{y}} p(\vec{y}) \log_2 \frac{1}{p(y_i)} = \sum_{i=1}^h \sum_{y_i=-1}^{+1} p(y_i) \log_2 \frac{1}{p(y_i)} = \sum_{i=1}^h S(y_i) .$$

Nel nostro caso $M(\vec{x}, \vec{y}) = S(\vec{y})$ e, a maggior ragione, $M(\vec{x}, y_i) = S(y_i)$ e dunque la distanza di Kullback Leibler ci viene

$$D_{KL}(p(\vec{y}), \prod_{i=1}^h p(y_i)) = \sum_{i=1}^h S(y_i) - S(\vec{y}) = \sum_{i=1}^h M(\vec{x}, y_i) - M(\vec{x}, \vec{y})$$

che però può essere minimizzata anche da una soluzione banale $M(\vec{x}, y_i) = M(\vec{x}, \vec{y}) = 0$ per cui bisogna aggiungere in generale il vincolo che $M(\vec{x}, \vec{y})$ assuma un certo valore. Nel caso del nostro strato di Perceptron è facile fissare $M(\vec{x}, y_i) = 1$, significa semplicemente che ogni Perceptron divide a metà gli input possibili e dunque ogni Perceptron porta 1 bit di informazione. Però in questo caso $\sum_{i=1}^h M(\vec{x}, y_i) = h$, che è anche un limite superiore alla capacità C e dunque vediamo che anche in questo caso minimizzare la distanza di Kullback Leibler $D_{KL}(p(\vec{y}), \prod_{i=1}^h p(y_i))$ equivale al principio Infomax.

A questo punto è chiaro che scopo dell'apprendimento è, per una data distribuzione degli input $p(\vec{x})$, raggiungere il valore massimo della mutua informazione, che nei casi fortunati, potrà essere uguale alla capacità (25). Per esempio nel semplice caso di input con distribuzione Gaussiana, si trova che il principio Infomax (con le ulteriori richieste che il codice sia invertibile e che i neuroni siano lineari) produce l'algoritmo che calcola le componenti principali degli input che dunque soddisfano, per uno strato di h neuroni lineari, tutti e tre i principi ora esposti. Chiaramente questo vale per $h \leq n$, cioè $\alpha \leq 1$, dato che non ci possono essere più di n componenti principali²³.

²³Nel caso $\alpha > 1$ le cose diventano più complicate e un analisi basata sulla meccanica statistica trova che, per una rete con input disposti lungo una dimensione, i pesi ottimali seguono la celebre curva a cappello messicano (vedi nei paragrafi 13.1.1 e 13.6).

Ricaviamo infine un algoritmo per un caso particolarmente semplice: un singolo neurone con le seguenti caratteristiche: input continuo x di distribuzione $p(x)$, ignota, output continuo e nessun rumore (che complicherebbe l'analisi anche se è indispensabile per un caso realistico), dunque:

$$y = \tanh(xw_1 + w_0)$$

che è una funzione continua e invertibile (e ricorda il caso delle cellule monopolari (LMC) dell'occhio della mosca). Cercheremo di massimizzare la mutua informazione di questa rete (Infomax). Abbiamo già visto che quando l'output è una funzione deterministica dell'input $M(x, y) = S(y)$ ma in questo caso y è una variabile continua per cui occorre usare l'entropia differenziale (59) che risulta

$$s(y) = \int_{-1}^1 p(y) \log_2 \frac{1}{p(y)} dx$$

ma la sua manipolazione richiede in generale maggiore attenzione dato che una variabile continua può portare infinita informazione. Noi aggireremo questi problemi ricordando che possiamo definire l'entropia come il valore di aspettazione dell'informazione (58) e cioè

$$s(y) = E[I(y)] = E \left[\log_2 \frac{1}{p(y)} \right] .$$

Anche se ignoriamo la distribuzione degli input $p(x)$ sappiamo che per la conservazione della probabilità deve valere

$$p(x)dx = p(y)dy$$

dalla quale si ricava la relazione generale di trasformazione delle distribuzioni per $y = f(x)$

$$p(y) = \frac{p(x)}{\left| \frac{dy}{dx} \right|}$$

dove il valore assoluto viene introdotto per non avere una densità di probabilità negativa ma nel nostro caso può essere ignorato dato che la derivata della nostra funzione di trasferimento, $\tanh(x)$, è sempre positiva. Possiamo ora calcolare la mutua informazione fra input e output

$$M(x, y) = s(y) = E[I(y)] = E \left[\log_2 \frac{1}{p(y)} \right]$$

e ancora, sostituendo la forma trovata per $p(y)$

$$M(x, y) = E \left[\log_2 \frac{1}{p(x)} \right] + E \left[\log_2 \left| \frac{dy}{dx} \right| \right] = s(x) + E \left[\log_2 \frac{dy}{dx} \right] .$$

Per massimizzare la mutua informazione di questa rete (Infomax) possiamo applicare un algoritmo di salita lungo il gradiente della funzione $M(x, y)$ per cui avremo una regola di aggiornamento dei pesi

$$\Delta w_i = \frac{\partial M(x, y)}{\partial w_i} = \frac{\partial E \left[\log_2 \frac{dy}{dx} \right]}{\partial w_i} = \log_2 e \frac{\partial E \left[\log \frac{dy}{dx} \right]}{\partial w_i}$$

dove il termine $s(x)$ è scomparso dato che chiaramente non dipende dai pesi e abbiamo cambiato base al logaritmo per facilitare le derivate. Visto che il valore di aspettazione rappresenta un integrale sugli x possiamo tranquillamente portare la derivazione rispetto ai pesi dentro il valore di aspettazione e ricavarne una regola di apprendimento da applicare al singolo input (apprendimento incrementale: anche la (11) può essere letta come un valore di aspettazione per input equiprobabili; inoltre omettiamo la costante $\log_2 e$)

$$\Delta w_i = \frac{\partial}{\partial w_i} \log \frac{dy}{dx} = \left(\frac{dy}{dx} \right)^{-1} \frac{\partial}{\partial w_i} \frac{dy}{dx}$$

e ricordando che per la nostra funzione di trasferimento $y = \tanh(x)$ la derivata vale $1 - y^2$ ricaviamo facilmente che nel nostro caso

$$\frac{dy}{dx} = \frac{d}{dx} \tanh(w_1 x + w_0) = (1 - y^2) w_1$$

e dunque

$$\left(\frac{dy}{dx} \right)^{-1} \frac{\partial}{\partial w_1} \frac{dy}{dx} = \frac{1}{(1 - y^2) w_1} [(1 - y^2) - w_1 2y(1 - y^2)x] = \frac{1}{w_1} - 2xy$$

e con simili conti per la soglia w_0 concludiamo

$$\begin{aligned} \Delta w_1 &= \frac{1}{w_1} - 2xy \\ \Delta w_0 &= -2y \end{aligned}$$

che formano l'algoritmo di Independent Component Analysis (ICA) introdotto da Bell e Sejnowski nel 1995 [2] e che generalizza l'analisi delle componenti principali (PCA) dato che in questo caso, per un output multidimensionale \vec{y} , le singole componenti y_i risultano indipendenti e non semplicemente a covarianza nulla.

Concludiamo questa parte osservando come l'applicazione delle idee della teoria dell'informazione ci ha permesso di trattare parecchi problemi che finora non sapevamo affrontare come per esempio di quanti neuroni abbiamo bisogno nello strato intermedio di una rete feed-forward (c in funzione di α , figura 42) e come costruire un algoritmo per l'apprendimento non supervisionato di uno strato di Perceptron (Infomax).

11 Memorie associative

Una delle prime applicazioni storiche delle reti neurali è quella delle memorie associative. Le trattiamo verso la fine visto che necessitano di matematica più complessa. Questi modelli sono stati proposti più volte in tempi e forme diverse: citiamo solo i lavori di Willshaw nel 1969 e di Hopfield nel 1982.

La memoria standard di un computer è logicamente organizzata in un certo numero di *parole*, formate da *bit*, ognuna delle quali è univocamente individuata da un numero: il suo *indirizzo*. Questo tipo di memoria ha le seguenti caratteristiche:

- non è affetta da errori: quello che si deposita in una parola si ritrova esattamente quando la si va a leggere;
- il contenuto di una parola può essere ritrovato solo se si sa l'indirizzo della parola stessa;
- è fragile nel senso che un solo bit sbagliato nell'indirizzo e/o nel contenuto da memorizzare non può essere corretto e può produrre errori irreparabili.

Al contrario la memoria biologica, anche se assai meno facile da definire esattamente, ha caratteristiche completamente diverse:

- non sempre quello che si è memorizzato può essere recuperato senza errori;
- il contenuto di una parola può essere ritrovato da *tutto* o *parte* del suo *contenuto*: è cioè reperibile per contenuto (e non v'è traccia di indirizzi);
- è robusta nel senso che esempi incompleti o affetti da errori permettono alla memoria di funzionare ugualmente.

Vediamo ora come con neuroni di McCulloch e Pitts si può organizzare una rete il cui comportamento richiama quello della memoria biologica, questo tipo di rete viene chiamato memoria associativa. Rispetto alle reti studiate finora qui prendiamo un approccio leggermente diverso: invece di studiare l'apprendimento ci concentreremo su quanto memorizzato dalla rete senza preoccuparci inizialmente di come sia avvenuto l'apprendimento.

Diversamente dal caso delle reti feed-forward ora non c'è una direzione preferenziale e tutti i neuroni possono essere connessi tra di loro come in figura 43. Consideriamo neuroni digitali con valore ± 1 e l' i -esimo neurone produrrà il suo output con la solita regola $y_i = \text{sgn}(\vec{x} \cdot \vec{w}_i)$. Il fatto che non ci sia più una direzione preferenziale rende inutile la differenza fra input e output e lo sottolineeremo indicandoli entrambi con S_i riservando il simbolo \vec{S} al generico stato della rete, ovviamente dati n neuroni ci sono 2^n diversi

stati possibili della rete. Per ogni neurone si ha dunque la regola per calcolare il suo stato

$$S'_i = \text{sgn}(\vec{w}_i \cdot \vec{S}) = \text{sgn} \left(\sum_{j=1}^n w_{ij} S_j \right) \quad (26)$$

dove abbiamo indicato con S'_i lo stato dell' i -esimo neurone *dopo* l'applicazione della regola e osserviamo come lo stato di ogni neurone influenzi tutti gli altri. È possibile immaginare casi nei quali la rete, anziché rimanere in uno stato, sia in perenne mutamento; per esercizio provate a disegnarne una fatta da tre soli neuroni.

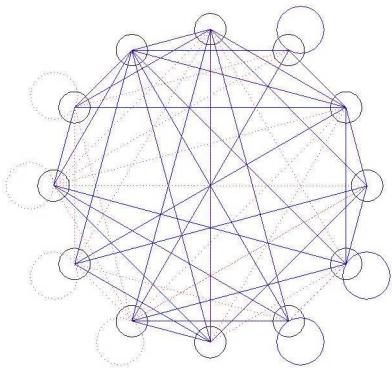


Figura 43: Rappresentazione schematica di una rete generale (ricorrente).

Questo tipo di reti può dunque mostrare dipendenza dal tempo: siamo in presenza di un *sistema dinamico* la cui evoluzione sarà in generale governata da equazioni differenziali. Vedremo che l'idea è supportata dal fatto che esiste un sistema fisico (un sistema di Ising, descritto nel paragrafo 12.1) che risulta estremamente simile a quello delle memorie associative. Per adesso però limitiamoci a prendere atto di questa possibilità e, assunto implicitamente che il tempo sia discretizzato (per cui da equazioni differenziali si passa a equazioni alle differenze finite), optiamo per una simulazione

di questo sistema nel quale i neuroni vengano tutti aggiornati con la (26). Pur con queste notevolissime semplificazioni rispetto ad un sistema dinamico reale, cioè *fisico*, restano ancora in sospeso parecchie domande una delle quali è: come si aggiornano i neuroni ? tutti insieme ? uno alla volta ? e, in questo caso, in che ordine ? Un'analisi più approfondita mostra come non ci siano differenze qualitative fra i diversi metodi e noi faremo riferimento al caso nel quale, scelto un neurone a caso, lo si aggiorna mantenendo fermi tutti gli altri. Questo modo di procedere, oltre a essere più facile da simulare al computer, ha anche il vantaggio di essere il più verosimile da un punto di vista biologico; ritorneremo su questo argomento nel paragrafo 12.3.

Ci resta ancora il problema di stabilire un modo per comunicare con la rete dall'esterno: sia per leggere un particolare stato (lettura) che per memorizzare uno stato e/o proporre il ritrovamento di un esempio memorizzato (scrittura). Trascureremo i dettagli di queste parti assumendo che lo stato \vec{S} possa essere letto dall'esterno e che, similmente, si possano forzare uno o più neuroni ad assumere un particolare stato.

Fissate le condizioni al contorno supponiamo ora di voler memorizzare un particolare stato $\vec{\xi}$ nella nostra rete: finora con ξ abbiamo indicato l'output

desiderato di un neurone, estendiamo la notazione allo stato di n neuroni che desideriamo sia memorizzato dalla rete. Una delle prime richieste da fare alla rete è che essa, se in questo stato, lo mantenga indefinitamente, chiediamo cioè che questo stato sia stabile per la nostra rete. Questa richiesta da:

$$\xi'_i = \operatorname{sgn}(\vec{w}_i \cdot \vec{\xi}) = \operatorname{sgn}\left(\sum_{j=1}^n w_{ij} \xi_j\right) \stackrel{?}{=} \xi_i$$

che, se soddisfatta, fa sì che nessun neurone cambi stato con l'applicazione della regola (26). Ricordando che $\xi_j = \pm 1$, ovviamente $\xi_j \xi_j = 1$ e dunque un'ovvia scelta per i pesi w_{ij} è sceglierli proporzionali a $\xi_i \xi_j$ cioè $w_{ij} = \gamma \xi_i \xi_j$ infatti allora

$$\xi'_i = \operatorname{sgn}\left(\sum_{j=1}^n w_{ij} \xi_j\right) = \operatorname{sgn}\left(\gamma \sum_{j=1}^n \xi_i \xi_j \xi_j\right) = \operatorname{sgn}\left(\gamma \xi_i \sum_{j=1}^n \xi_j \xi_j\right) = \operatorname{sgn}(\gamma \xi_i n)$$

e, fissando dunque la costante di proporzionalità $\gamma = \frac{1}{n}$, abbiamo trovato che per memorizzare lo stato $\vec{\xi}$ i pesi della rete debbono prendere i valori

$$w_{ij} = \frac{1}{n} \xi_i \xi_j \quad (27)$$

con i quali si ricava

$$\xi'_i = \operatorname{sgn}(\vec{w}_i \cdot \vec{\xi}) = \operatorname{sgn}(\xi_i) = \xi_i$$

e dunque l'esempio $\vec{\xi}$ è uno stato stabile per la rete.

La stabilità dell'esempio memorizzato si riferisce alla parte *statica*; in questo caso molto semplice si può capire cosa succede anche per la *dinamica*: per farlo supponiamo che lo stato iniziale della rete sia $\vec{S} \neq \vec{\xi}$, avremo allora che l'applicazione della regola (26) darà

$$S'_i = \operatorname{sgn}\left(\sum_{j=1}^n w_{ij} S_j\right) = \operatorname{sgn}\left(\frac{1}{n} \sum_{j=1}^n \xi_i \xi_j S_j\right) = \operatorname{sgn}\left(\frac{\xi_i}{n} \sum_{j=1}^n \xi_j S_j\right) = \operatorname{sgn}\left(\frac{\xi_i}{n} \vec{\xi} \cdot \vec{S}\right)$$

Per procedere osserviamo che visto che i vettori \vec{S} ed $\vec{\xi}$ hanno tutte le componenti che valgono ± 1 ovviamente sarà

$$-n \leq \vec{\xi} \cdot \vec{S} \leq n$$

e volendo essere più precisi possiamo osservare che se \vec{S} e $\vec{\xi}$ hanno k componenti uguali, e, necessariamente, $n - k$ componenti diverse, allora

$$\vec{\xi} \cdot \vec{S} = k - (n - k) = 2k - n$$

e sostituendo nella relazione precedente troviamo

$$S'_i = \operatorname{sgn} \left(\xi_i \frac{2k - n}{n} \right)$$

per cui il risultato cambia a seconda del numero k di componenti uguali fra \vec{S} e $\vec{\xi}$

$$S'_i = \begin{cases} \xi_i & \text{se } k > \frac{n}{2} \\ -\xi_i & \text{altrimenti} \end{cases}$$

questa relazione mostra che se lo stato \vec{S} è “abbastanza vicino” ($k > \frac{n}{2}$) a $\vec{\xi}$ ogni applicazione della regola porta a uno stato \vec{S}' che ha un numero di componenti $k' \geq k > \frac{n}{2}$ e dunque più “vicino” a $\vec{\xi}$. Ad ogni aggiornamento il numero di bit uguali può solo aumentare finché, dopo un numero sufficientemente grande di aggiornamenti, coinciderà con $\vec{\xi}$ dove poi resta stabile: nel linguaggio dei sistemi dinamici si dice che lo stato $\vec{\xi}$ è un attrattore del sistema. Se invece lo stato iniziale \vec{S} era “distante” ($k < \frac{n}{2}$) da $\vec{\xi}$ si troverà necessariamente $k' \leq k < \frac{n}{2}$ cioè sempre più lontano da $\vec{\xi}$ ma più vicino a $-\vec{\xi}$ che sarà un altro attrattore del sistema.

In altre parole abbiamo mostrato che lo spazio delle configurazioni possibili della rete è ripartito in due parti uguali i cui punti hanno rispettivamente $\vec{\xi}$ e $-\vec{\xi}$ come attrattori; questi due insiemi di punti sono i cosiddetti *bacini di attrazione* di $\vec{\xi}$ e $-\vec{\xi}$, la situazione è schematizzata in figura 44²⁴.

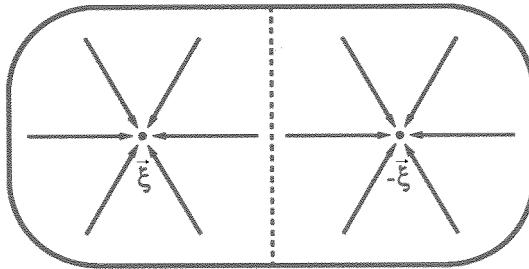


Figura 44: Bacini di attrazione nel caso di un solo esempio memorizzato.

Abbiamo dunque scoperto che questa rete, con i pesi dati dalla (27), può memorizzare l'esempio $\vec{\xi}$ che risulta stabile e, dal punto di vista della dinamica, è al centro di un bacino di attrazione grande come metà dello spazio delle configurazioni. Inoltre la situazione è simmetrica per l'esempio $-\vec{\xi}$, in termini fotografici, il negativo dell'esempio dato. Dunque, restando nell'esempio delle immagini, se memorizzo un'immagine binaria in una rete di questo tipo e ripropongo successivamente alla rete una versione rumorosa o parziale della stessa immagine sarà in grado di ritrovare l'immagine originale. Da un suggerimento parziale sono riuscito a ritrovare l'immagine memorizzata: si capisce l'origine del nome di memoria associativa, un esempio fatto con le immagini si trova in figura 45.

Prima di procedere osserviamo quanto ci è costerebbe costruire una memoria di questo genere: per farla avremmo bisogno di n bit per lo stato dei neuroni e n^2 bit per il valori dei pesi, che a parte la costante di proporzionalità $\frac{1}{n}$, possono assumere solo i valori ± 1 . In totale $n(n + 1)$ bit per

²⁴In realtà lo spazio delle configurazioni è un ipercubo di vertici $\{\pm 1\}^n \in \mathbb{R}^n$ nel quale $\vec{\xi}$ e $-\vec{\xi}$ stanno su due vertici opposti e i due bacini di attrazione sono separati dall'iperpiano $\vec{x} \cdot \vec{\xi} = 0$.



Figura 45: Esempio di come una memoria associativa trova delle immagini memorizzate di 180×130 pixels. Nella colonna di sinistra c'è lo stato iniziale, in quella di destra lo stato fisso finale.

memorizzarne solo n però in cambio abbiamo ottenuto che questa memoria è in grado di ricostruire immagini rumorose o parziali. In più si vede che l'esempio è, in pratica, memorizzato nei pesi e si può intuire che un eventuale peso sbagliato o mancante non sarà sufficiente a far cambiare segno alla (26) per cui questa memoria è anche insensibile (almeno fino a un certo livello) alla variazione dei pesi. In altre parole modificare qualche bit di pesi permette alla memoria di funzionare ancora.

Veniamo ora al caso, più interessante, nel quale si vogliono memorizzare m esempi $\vec{\xi}_\nu$, generalizzando ovviamente la (27) definiamo i pesi con

$$w_{ij} = \frac{1}{n} \sum_{\nu=1}^m \xi_{i\nu} \xi_{j\nu} \quad (28)$$

che è un'evidente applicazione della regola di Hebb estesa! Iniziamo anche qui verificando la stabilità di un esempio $\vec{\xi}_\mu$ (la stabilità assumerà un significato più ampio nel paragrafo 11.4)

$$\xi_{i\mu} \stackrel{?}{=} \text{sgn} (\vec{w}_i \cdot \vec{\xi}_\mu) = \text{sgn} \left(\sum_{j=1}^n w_{ij} \xi_{j\mu} \right) = \text{sgn} \left(\frac{1}{n} \sum_{j=1}^n \sum_{\nu=1}^m \xi_{i\nu} \xi_{j\nu} \xi_{j\mu} \right) = \quad (29)$$

e se nella seconda somma isoliamo il termine con $\nu = \mu$ facilmente troviamo

$$= \operatorname{sgn} \left(\frac{1}{n} \sum_{j=1}^n \xi_{i\mu} \xi_{j\mu} \xi_{j\mu} + \frac{1}{n} \sum_{j=1}^n \sum_{\nu \neq \mu} \xi_{i\nu} \xi_{j\nu} \xi_{j\mu} \right) = \operatorname{sgn} \left(\xi_{i\mu} + \frac{1}{n} \sum_{j=1}^n \sum_{\nu \neq \mu} \xi_{i\nu} \xi_{j\nu} \xi_{j\mu} \right) =$$

che, ricordando che $\xi_{i\mu} \xi_{i\mu} = 1$, possiamo scrivere come

$$= \operatorname{sgn} \left(\xi_{i\mu} \left(1 + \frac{\xi_{i\mu}}{n} \sum_{j=1}^n \sum_{\nu \neq \mu} \xi_{i\nu} \xi_{j\nu} \xi_{j\mu} \right) \right)$$

esaminando la quale osserviamo che $\xi_{i\mu}$ è stabile o meno a seconda del segno del fattore in parentesi. Dunque possiamo dire che *non* vi sarà stabilità se vale

$$\xi_{i\mu} \sum_{j=1}^n \sum_{\nu \neq \mu} \xi_{i\nu} \xi_{j\nu} \xi_{j\mu} < -n . \quad (30)$$

Questa somma contiene $n(m-1)$ termini ognuno dei quali vale ± 1 e per poter sapere se la condizione è rispettata dovremmo conoscere esattamente tutti gli esempi memorizzati, il che chiaramente non è pratico.

Possiamo però formulare questo risultato in forma probabilistica assumendo che i singoli termini della somma siano variabili aleatorie e supponiamo di conoscere soltanto la probabilità p che uno degli addendi valga 1 (e $1-p$ che valga -1); potremo allora calcolare la *probabilità* che sia soddisfatta la (30).

Possiamo dunque concludere che la probabilità P_I che vi sia una condizione di instabilità per un bit sarà uguale alla probabilità che sia soddisfatta la (30). Prima di procedere una breve ripetizione del problema del cammino casuale unidimensionale.

11.1 Il cammino casuale unidimensionale

Questo problema, noto in inglese come *random walk*, è il problema di calcolare la distanza dall'origine che raggiunge un uomo che, partendo dall'origine, lancia N volte una moneta che da testa con probabilità p , e per ogni testa fa un passo avanti e per ogni croce fa un passo indietro. Sappiamo che la distribuzione del numero di teste in N lanci segue la distribuzione binomiale, ripetuta nell'appendice A.2.

Dopo N lanci il tale avrà fatto k passi avanti e $N-k$ passi indietro per cui si troverà a una distanza x dall'origine

$$x = k - (N - k) = 2k - N$$

osserviamo che se k è distribuita in modo binomiale anche la variabile x sarà distribuita secondo una binomiale e possiamo calcolarne il valore medio

$$E[x] = E[2k - N] = 2E[k] - N = 2Np - N = N(2p - 1) \quad (31)$$

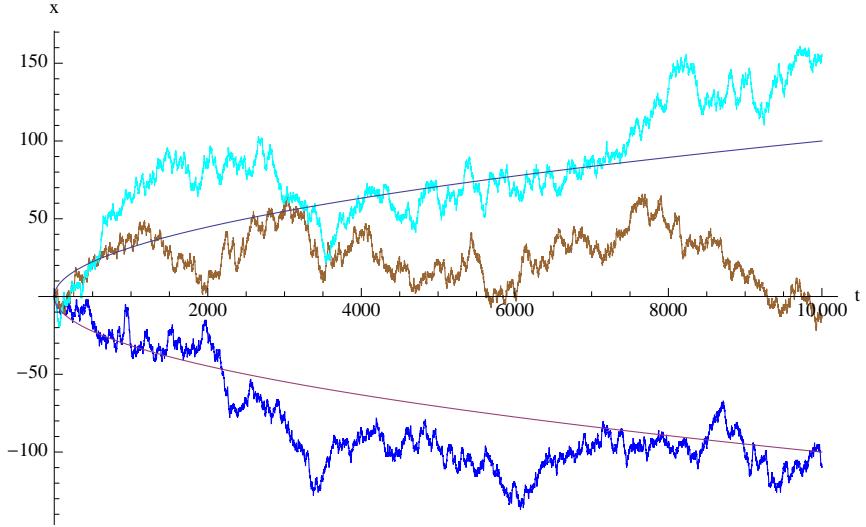


Figura 46: 3 random walk di 10^4 passi con $p = 1/2$, la curva continua è $\pm\sqrt{N}$.

e per una moneta equa, $p = 1/2$, ovviamente, $E[x] = 0$. Un risultato meno intuitivo si ottiene calcolando la varianza di x che risulta

$$\begin{aligned}\text{Var}[x] &= E[x^2] - E[x]^2 = 4E[k^2] - 4NE[k] + N^2 - 4N^2p^2 + 4N^2p - N^2 = \\ &= 4(E[k^2] - N^2p^2)\end{aligned}$$

e riconosciamo, in parentesi, la varianza di k per cui

$$\text{Var}[x] = 4Np(1 - p) \quad (32)$$

che, per la solita moneta equa, $p = 1/2$, vale $\text{Var}[x] = N$ o, se si preferisce, lo scarto quadratico medio di x vale \sqrt{N} . Questo ci dice che al crescere del numero di lanci l'uomo si allontana sempre di più dall'origine in corrispondenza a lunghe serie di teste o croci ma in media non si sposta, in figura 46 ci sono tre esempi di cammini casuali di 10.000 passi. In altre parole ci si può aspettare delle serie di teste dell'ordine \sqrt{N} , ricordatevene se giocate a testa e croce (in questo caso la strategia vincente è raddoppiare sempre la posta... occorre però una quantità esponenziale di soldi...).

Questi risultati si possono anche ottenere facilmente osservando che x è la somma di N variabili casuali y_i ognuna delle quali vale ± 1 , ha valor medio $E[y_i] = 2p - 1$ e $E[y_i^2] = 1$ e dunque $\text{Var}[y_i] = 1 - (2p - 1)^2 = 4p(1 - p)$.

In sostanza dunque x è una quantità che segue una distribuzione binomiale e dunque, quando il numero di lanci $N \rightarrow \infty$, potremo approssimare la distribuzione degli x con una Gaussiana $G_{\mu\sigma}$ (52) di parametri $\mu = N(2p - 1)$ e $\sigma = 2\sqrt{Np(1 - p)}$.

11.2 Riprendiamo il conto sospeso

Dovendo verificare le condizioni per le quali vale la (30) osserviamo che essa è la somma di $n(m-1)$ addendi ognuno dei quali vale ± 1 con probabilità p e $1-p$. Ovviamente la distribuzione degli $n(m-1)$ termini della somma segue una distribuzione binomiale (appendice A.2) di parametri $N = n(m-1)$ e p . La somma di questi $n(m-1)$ termini è un caso del cammino casuale unidimensionale appena studiato. Se nella somma x ci sono esattamente k addendi che valgono 1 la somma vale $x = 2k - n(m-1)$ inoltre, come abbiamo visto, la somma segue una distribuzione binomiale che ha i parametri trovati per il cammino casuale.

La probabilità che valga la (30) è la somma delle probabilità che la somma prenda uno qualsiasi dei valori possibili compresi fra $-n(m-1)$ e $-n-1$, come rappresentato graficamente in figura 47. Il fattore $\xi_{i\mu}$ che la precede rompe un po' perché scambiando $i+1$ con -1 scambia p con $1-p$ nel conto finale. Questo effetto però non c'è se $p = 1/2$ dato che allora $p = 1-p$.

Per poter portare a termine il conto occorre mettersi precisamente in questa condizione introducendo una *semplificazione*: supponiamo che gli esempi $\vec{\xi}_\nu$ abbiano una distribuzione $P(\vec{\xi})$ completamente fattorizzata e che i singoli bit abbiano tutti probabilità $1/2$ di valere 1 e cioè:

$$P(\vec{\xi}) = p(\xi_1)p(\xi_2) \cdots p(\xi_n) \quad \text{e anche} \quad p(\xi_i = 1) = \frac{1}{2} \quad \forall i = 1, 2, \dots, n \quad (33)$$

che significa che tutti i bit di tutti gli esempi sono casuali e indipendenti, cioè ottenuti scegliendo, con probabilità $1/2$, fra +1 in -1 per ogni coordinata. Da questa ipotesi discende che i vari termini della (30) assumono valore +1 (e -1) con probabilità $p = 1/2$ (dimostratelo per esercizio, basta considerare tutti i casi possibili) e discende pure che tutti i bit degli esempi risultano completamente indipendenti fra loro sia all'interno dello stesso esempio che fra esempi diversi (la distribuzione è fattorizzata)²⁵. Al momento sembra un ipotesi molto forte ma vedremo che non è così male come sembra. Con questa ipotesi il fattore $\xi_{i\mu}$ della (30) possiamo dimenticarlo dato che scambiare p con $1-p$ non ha effetto se $p = 1/2$. Potremo ora finalmente calcolare la

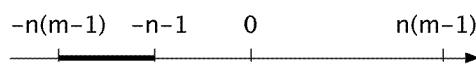


Figura 47: Range di valori possibili per la somma in (30) che soddisfano la diseguaglianza.

²⁵generalizzando la (61) si trova

$$\mathcal{S}(\vec{\xi}) \leq \sum_{i=1}^n \mathcal{S}(\xi_i) \leq n$$

e con le ipotesi sulla distribuzione $P(\vec{\xi})$ si deduce facilmente che siamo nel caso $\mathcal{S}(\vec{\xi}) = n$ in cui ogni esempio $\vec{\xi}_\nu$ ha esattamente n bit di informazione.

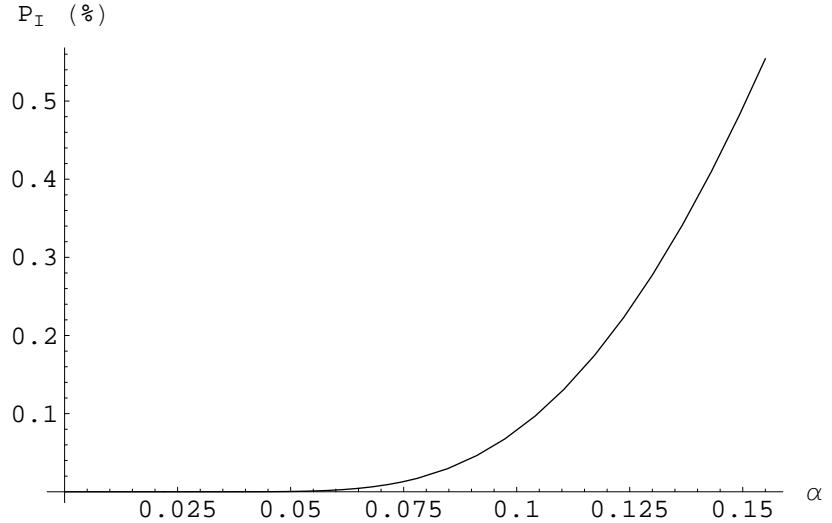


Figura 48: Probabilità P_I (in %) di avere instabilità su un bit al variare del rapporto $\alpha = \frac{m}{n}$ dalla (34).

probabilità di instabilità di un bit P_I come

$$P_I = \sum_{x=-n-1}^{-n(m-1)} \text{Prob}\left(\sum_{j=1}^n \sum_{\nu \neq \mu} \xi_{i\nu} \xi_{j\nu} \xi_{j\mu} = x\right)$$

Supponiamo ora di voler fare questo conto nel limite di $n, m \rightarrow \infty$; in questo caso possiamo approssimare la somma di probabilità binomiali con un integrale della Gaussiana $G_{\mu\sigma}$ che approssima la binomiale e avremo

$$P_I \simeq \int_{-n(m-1)-\frac{1}{2}}^{-n-\frac{1}{2}} G_{\mu\sigma}(t) dt$$

Per trovare i parametri della Gaussiana approssimante per prima cosa osserviamo che stiamo considerando un cammino casuale unidimensionale con $p = 1/2$ per cui il valor medio è 0 e la varianza N , che nel nostro caso corrisponde al numero di addendi $N = n(m-1)$. Per $n, m \rightarrow \infty$ con $\alpha := \frac{m}{n} \rightarrow$ costante ovviamente $n(m-1) \simeq nm$ e per il limite di integrazione $-n-1/2 \simeq -n$. Dunque i parametri da usare nella Gaussiana approssimante risultano

$$\mu = 0 \quad \sigma = \sqrt{nm}$$

Infine con la (53) che appare in appendice A.2 ricaviamo facilmente

$$P_I \simeq \int_{-\infty}^{-n} G_{0\sqrt{nm}}(t) dt = \frac{1}{2} \left[1 - \text{erf}\left(\frac{1}{\sqrt{2\alpha}}\right) \right] \quad (34)$$

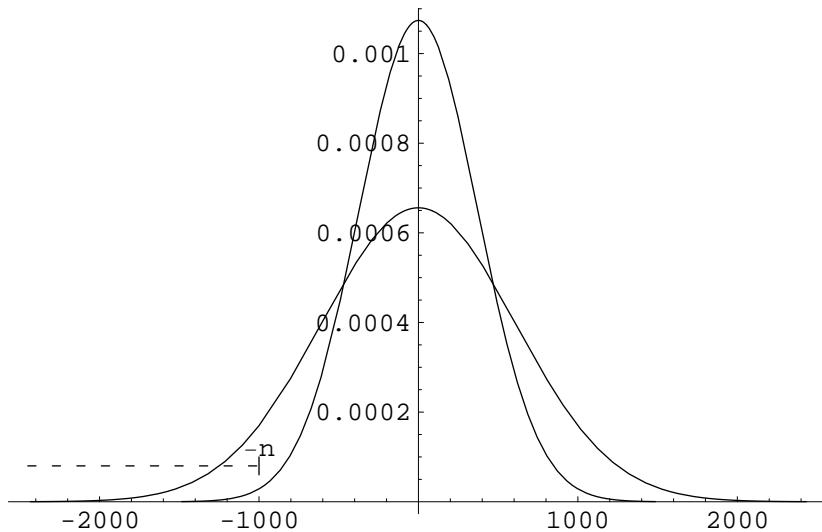


Figura 49: Due casi della Gaussiana che appare nella (34) calcolata per $n = 1000$ e per due valori di α , rispettivamente 0.138 e 0.370; in entrambi i casi l'integrale va da $-\infty$ a $-n$. Si vede che a un maggior valore di α corrisponde una distribuzione più larga e di conseguenza una maggiore probabilità di instabilità P_I .

che mostra che P_I dipende solo dal rapporto fra numero di neuroni n e numero di esempi m e il suo andamento si trova in figura 48. Notiamo che, dato che l'integrale si estende fino a $-n$, per una Gaussiana che è centrata in 0, in ogni caso avremo che $P_I < 1/2$.

Per capire meglio questo risultato conviene pensare graficamente a questo integrale come fatto in figura 49, si vede che al crescere del numero degli esempi memorizzati, cresce $\sigma = \sqrt{nm}$ e perciò l'integrale che calcola P_I aumenta: più esempi memorizziamo nella rete maggiore è la probabilità che un bit diventi instabile. Possiamo ragionare alla rovescia e, fissato un limite accettabile di probabilità di instabilità per un bit P_I , ricavare il valore corrispondente di α e con esso il numero massimo di esempi che posso memorizzare nella rete. Nella tabella 2 abbiamo riportato il massimo valore di α in corrispondenza a vari livelli di probabilità di errore.

Calcolata l'instabilità su un singolo bit ora una richiesta più stringente potrebbe essere che tutti gli n bit di un esempio siano stabili con una certa probabilità, per esempio all'1%, cioè

$$P_S^n = (1 - P_I)^n > \frac{99}{100}$$

$P_I(\%)$	α
0.1	0.105
0.36	0.138
1	0.185
5	0.37
10	0.61

Tabella 2: P_I (in %) e corrispondenti α .

e sviluppando la potenza e tenendo solo il primo ordine in P_I , dato che è un numero piccolo, si ricava

$$P_I < \frac{1}{100n}$$

e in questo caso, come era logico aspettarsi, il limite su P_I cala al crescere di n dato che vogliamo tutti gli n bit stabili. Questo implica che per rispettare questo limite devo far sì che $\alpha = \frac{m}{n} \rightarrow 0$ quando $n \rightarrow \infty$, cioè $\sigma/n \rightarrow 0$. Dato che P_I dipende solo da α possiamo usare la (34) per trasformare la diseguaglianza in un limite su α ; sviluppando in serie P_I nell'intorno di 0 al primo ordine in α si trova

$$P_I \simeq \sqrt{\frac{\alpha}{2\pi}} e^{-\frac{1}{2\alpha}}$$

prendendo il logaritmo della quale e trascurando i termini costanti e in $\log \alpha$ facilmente si trova che, per rispettare la richiesta di stabilità su tutti gli n bit, deve valere

$$\alpha < \frac{1}{2 \log 100n}$$

Se poi addirittura voglio che tutti i bit di tutti gli m esempi memorizzati siano stabili, sempre all'1%, basta sostituire $mn = \alpha n^2$ al posto di n nelle relazioni precedenti e dunque in questo caso (trascurando il termine $2 \log \alpha$)

$$\alpha < \frac{1}{4 \log 100n}$$

Dobbiamo notare che il calcolo appena fatto per P_I mi da la probabilità di instabilità di uno o più bit ma sarà solo un componente del calcolo, in realtà assai più complesso, che l'esempio sia stabile non solo staticamente, come abbiamo or ora fatto, ma *dinamicamente*. Supponiamo di mettere il sistema nello stato esatto $\vec{\xi}_\mu$ e di lasciarlo evolvere con la (26); abbiamo visto che ci possiamo aspettare che $n P_I$ bit cambino il loro stato allontanando il sistema dallo stato di partenza $\vec{S} = \vec{\xi}_\mu$. A questo punto potrebbe succedere che dopo un primo bit di instabilità ne arrivi un secondo e poi un terzo e via via ci si allontani dal punto desiderato di stabilità $\vec{\xi}_\mu$. Studiando questo fenomeno, vedi paragrafo 12.4, si trova che se $\alpha > 0.138$ non c'è più alcun effetto di memoria, cioè non c'è stabilità dinamica della memoria. Dalla tabella 2 vediamo che questo corrisponde alla probabilità di instabilità sul singolo bit del 0.36%. In realtà, con conti più sofisticati, si trova che, al raggiungimento della stabilità dinamica, circa l'1.6% dei bit saranno sbagliati.

11.3 Osservazioni sparse sulla matrice dei pesi

Osserviamo che possiamo mettere i pesi w_{ij} definiti dalla (28) in una matrice quadrata di ordine n , W e salta agli occhi che è simmetrica dato che $w_{ij} =$

w_{ji} . Mettendo gli m esempi in una matrice X di n righe ed m colonne allora la (28) si scrive $W = \frac{1}{n}XX^T$ che, a parte un fattore costante, coincide con la matrice di covarianza degli esempi (54) e dunque $W = \alpha C$.

I pesi simmetrici rappresentano un ipotesi assai poco biologica; inoltre si vede che risultano definiti anche i pesi w_{ii} che danno la sinapsi di un neurone con se stesso, sono i pesi che stanno sulla diagonale della matrice dei pesi W e facilmente si vede che valgono tutti

$$w_{ii} = \frac{m}{n} = \alpha > 0$$

e tendono ad introdurre nella dinamica della rete delle soluzioni spurie, cioè punti di attrazione non corrispondenti ad alcuno degli esempi memorizzati. Infatti scrivendo $S'_i = \text{sgn}(w_{ii}S_i + \sum_{j \neq i} w_{ij}S_j)$ osserviamo che se $|\sum_{j \neq i} w_{ij}S_j| < w_{ii} = \alpha$ allora si perde completamente l'effetto dovuto agli esempi memorizzati e sono stabili sia $S_i = 1$ che $S_i = -1$. Per ridurre questo effetto, quasi sempre, nella simulazione di memorie associative si usa imporre $w_{ii} = 0$.

★ ★ ★

Osserviamo che se vale l'ipotesi di fattorizzazione della distribuzione degli esempi (33) ne discende che tutti gli elementi non diagonali della matrice dei pesi W sono variabili casuali che sono rappresentate da cammini casuali unidimensionali con $p = 1/2$ e che dunque hanno valore di aspettazione $E[w_{ij}] = 0$ e varianza $\frac{m}{n^2}$; per n grande sono dunque ben approssimati da una distribuzione Gaussiana con questi parametri. Risulta utile definire anche la matrice degli overlap $Q = \frac{1}{n}X^TX$, quadrata di ordine m , il cui generico elemento vale

$$q_{\mu\nu} = \frac{1}{n} \vec{\xi}_\mu \cdot \vec{\xi}_\nu \quad -1 \leq q_{\mu\nu} \leq 1 \quad q_{\mu\mu} = 1$$

e anche questa matrice, nell'ipotesi di fattorizzazione della distribuzione degli esempi (33), ha elementi non diagonali casuali con valore di aspettazione $E[q_{\mu\nu}] = 0$ e varianza $\frac{1}{n}$.

★ ★ ★

Un caso interessante è quello di considerare degli esempi strettamente *ortogonali* fra di loro e cioè

$$\frac{1}{n} \vec{\xi}_\mu \cdot \vec{\xi}_\nu = \frac{1}{n} \sum_{j=1}^n \xi_{j\mu} \xi_{j\nu} = \delta_{\mu\nu} = Q \quad (\text{n pari})$$

in questo caso con la (30) si dimostra facilmente (fatelo) che tutti i bit di tutti gli esempi sono diventati stabili. Perdipiù visto che per le proprietà dello

spazio \mathbb{R}^n ce ne potrebbero essere fino a $m = n$ ortogonali fra loro (che non è certo si possa sempre realizzare... [congettura di Hadamard](#)) e sembrerebbe che la memoria possa dunque arrivare fino ad $\alpha = 1$. Notiamo che per $m = n$ esempi la matrice X è quadrata di ordine n e la relazione di ortonormalità fra esempi appena vista si scrive $X^T X = n\mathbb{1}$, dalla quale facilmente segue (basta moltiplicare questa relazione per X e usare la proprietà associativa) che anche $XX^T = n\mathbb{1}$ ma, ricordando che la matrice dei pesi si può scrivere

$$W = \frac{1}{n} XX^T = \mathbb{1} \quad (35)$$

troviamo che la matrice dei pesi corrisponde alla matrice identità e ogni stato risulta stabile dato che

$$S'_i = \text{sgn}(\vec{w}_i \cdot \vec{S}) = \text{sgn}(S_i)$$

ma il risultato è poco interessante dato che ogni neurone risulta connesso solo con se stesso. In pratica in questo caso tutti i 2^n stati sono stabili: ogni stato che venga messo nella memoria rimane stabile e la memoria associativa ha smesso di funzionare.

Si può evitare questa soluzione banale memorizzando solo $m < n$ esempi; in questo caso X ha dimensioni $n \times m$ e si trova che se la matrice dei pesi viene definita con l'inversa generalizzata²⁶, cioè

$$W = \frac{1}{n} X Q^{-1} X^T = X(X^T X)^{-1} X^T$$

dalla quale segue facilmente $WX = X$ cioè, esempio per esempio:

$$W\vec{\xi}_\mu = \vec{\xi}_\mu$$

e tutti gli m esempi risultano stabili e la memoria associativa ha ripreso a funzionare.

* * *

Proponiamo ora qualche accorgimento per migliorare il funzionamento della memoria associativa specialmente quando si desiderino memorizzare degli esempi con correlazioni come per esempio delle immagini per le quali l'ipotesi di fattorizzazione della distribuzione (33) è paleamente violata nel qual caso la memoria funziona solo con pochissimi esempi memorizzati.

Per prima cosa vediamo come migliorare il valore delle sinapsi per rendere almeno stabili gli esempi memorizzati. Ricordiamo che la condizione di stabilità per un esempio è

$$\xi_{i\mu} \stackrel{?}{=} \text{sgn}(\vec{w}_i \cdot \vec{\xi}_\mu)$$

²⁶si definisce inversa generalizzata o [pseudo inversa](#) una matrice che gode di solo alcune delle proprietà dell'inversa, ad esempio la pseudo-inversa della nostra X è la matrice $X^* := (X^T X)^{-1} X^T$ di dimensioni $m \times n$ che gode della proprietà $X^* X = \mathbb{1}_m$.

che possiamo riscrivere

$$\xi_{i\mu} \vec{w}_i \cdot \vec{\xi}_\mu \begin{cases} > 0 & \Rightarrow \text{stabile} \\ < 0 & \Rightarrow \text{instabile} \end{cases}$$

dato che quello che conta sono i segni di $\xi_{i\mu}$ e $\vec{w}_i \cdot \vec{\xi}_\mu$. Osserviamo ora che con questa quantità possiamo controllare non solo la stabilità dello stato $\vec{\xi}_\mu$ ma anche la dinamica del sistema nelle vicinanze di $\vec{\xi}_\mu$. Per vederlo iniziamo osservando che, vista la definizione dei pesi (28), per ogni stato \vec{S} della memoria $-m \leq \vec{w}_i \cdot \vec{S} \leq m$. Sia ora V_μ l'insieme degli n stati 'vicini' a $\vec{\xi}_\mu$ nel senso che hanno solo un bit diverso da questi. Per ogni $\vec{S}' \in V_\mu$ vale

$$\vec{w}_i \cdot \vec{S}' = \vec{w}_i \cdot \vec{\xi}_\mu \pm 2w_{il} \quad -\alpha \leq w_{il} \leq \alpha \quad \forall l$$

se \vec{S}' e $\vec{\xi}_\mu$ differiscono nella l -esima coordinata e di conseguenza

$$\xi_{i\mu} \vec{w}_i \cdot \vec{S}' = \xi_{i\mu} \vec{w}_i \cdot \vec{\xi}_\mu \pm 2w_{il}$$

e se $\xi_{i\mu} \vec{w}_i \cdot \vec{\xi}_\mu > 2\alpha$, qualsiasi siano il valore e il segno di w_{il} varrà sempre

$$\xi_{i\mu} \vec{w}_i \cdot \vec{S}' > 0$$

il che significa che tutti gli stati $\vec{S}' \in V_\mu$, all'evoluzione, possono solo avvicinarsi a $\vec{\xi}_\mu$ (se la differenza era nel i -esima coordinata). In altre parole rinforzando la richiesta iniziale con

$$\xi_{i\mu} \vec{w}_i \cdot \vec{\xi}_\mu > 2k\alpha > 0$$

posso anche regolare la dinamica per tutti gli stati che distano al più k bit da $\vec{\xi}_\mu$. Ricordiamo che per un solo esempio memorizzato valeva addirittura $\xi_{i\mu} \vec{w}_i \cdot \vec{\xi}_\mu = 1 = \frac{n}{2}2\alpha = m = 1$ e iniziamo proprio con questa richiesta piuttosto forte

$$\xi_{i\mu} \vec{w}_i \cdot \vec{\xi}_\mu = 1 > 0 .$$

Ora mostriamo che, partendo dalle sinapsi della regola di Hebb standard (28), se presentiamo l'esempio $\vec{\xi}_\mu$ alla rete e poi le aggiorniamo con la regola

$$w'_{ij} = w_{ij} + \Delta w_{ij} = w_{ij} + \frac{1}{n}(1 - \xi_{i\mu} \vec{w}_i \cdot \vec{\xi}_\mu) \xi_{i\mu} \xi_{j\mu}$$

che in sostanza "rinforza" la sinapsi w_{ij} con il valore $\xi_{i\mu} \xi_{j\mu}$ a meno che la condizione di stabilità $\xi_{i\mu} \vec{w}_i \cdot \vec{\xi}_\mu = 1$ non sia già soddisfatta. Dopo questo apprendimento la stabilità per $\vec{\xi}_\mu$ vale

$$\begin{aligned} \xi_{i\mu} \vec{w}'_i \cdot \vec{\xi}_\mu &= \xi_{i\mu} \vec{w}_i \cdot \vec{\xi}_\mu + \xi_{i\mu} \sum_{j=1}^n \Delta w_{ij} \xi_{j\mu} = \xi_{i\mu} \vec{w}_i \cdot \vec{\xi}_\mu + \xi_{i\mu} \frac{1}{n} \sum_{j=1}^n (1 - \xi_{i\mu} \vec{w}_i \cdot \vec{\xi}_\mu) \xi_{i\mu} \xi_{j\mu} \xi_{j\mu} = \\ &= \xi_{i\mu} \vec{w}_i \cdot \vec{\xi}_\mu + \xi_{i\mu}^2 (1 - \xi_{i\mu} \vec{w}_i \cdot \vec{\xi}_\mu) \frac{1}{n} \sum_{j=1}^n \xi_{j\mu}^2 = \xi_{i\mu} \vec{w}_i \cdot \vec{\xi}_\mu + (1 - \xi_{i\mu} \vec{w}_i \cdot \vec{\xi}_\mu) = \\ &= 1 . \end{aligned}$$

Questo aggiornamento delle sinapsi ha reso perfettamente stabile l'esempio $\vec{\xi}_\mu$. A questo punto si può procedere allo stesso modo con un altro esempio. Chiaramente facendo l'aggiornamento per un altro esempio $\vec{\xi}_\nu$, viene spontaneo chiedersi se vale ancora la stabilità per $\vec{\xi}_\mu$. Non è difficile rispondere a questa domanda e si trova, che procedendo con questi aggiornamenti all'infinito, la matrice dei pesi converge a:

$$W \rightarrow \frac{1}{n} X Q^{-1} X^T$$

che è proprio la soluzione della pseudoinversa incontrata poco fa. Con questa regola abbiamo trovato che, rinforzando gli esempi memorizzati con la regola di Hebb, si arriva ad una matrice dei pesi che, perlomeno, garantisce la stabilità di tutti gli esempi memorizzati.

Se siamo meno ingordi e chiediamo che per gli esempi sia richiesta una condizione di stabilità meno forte e precisamente

$$\xi_{i\mu} \vec{w}_i \cdot \vec{\xi}_\mu > 2k\alpha$$

in questo caso la regola di aggiornamento dei pesi diventa

$$w'_{ij} = w_{ij} + \Delta w_{ij} = w_{ij} + \frac{1}{n} \Theta \left(2k\alpha - \frac{\xi_{i\mu} \vec{w}_i \cdot \vec{\xi}_\mu}{\sqrt{\sum_{ij} w_{ij}^2}} \right) (1 - \delta_{ij}) \xi_{i\mu} \xi_{j\mu}$$

dove:

- la funzione $\Theta(\dots)$ garantisce che ci sia apprendimento solo quando la condizione di stabilità non è rispettata;
- è comparso un fattore di normalizzazione per tenere sotto controllo la lunghezza dei vettori dei pesi;
- il fattore $(1 - \delta_{ij})$ impedisce l'apprendimento sui termini diagonali w_{ii} .

Con queste modifiche l'algoritmo migliora di molto e si può dimostrare che converge rapidamente verso una matrice dei pesi che rende stabili tutti gli eventi memorizzati e, più precisamente, lo fa in un numero *finito* di passi; inoltre il valore di $2k\alpha > 0$ regola le dimensioni dei bacini di attrazione degli esempi (maggiori dettagli nel paragrafo 9.1.3 del libro di Müller e Reinhardt [10]).

★ ★ ★

Una volta resi stabili tutti gli esempi memorizzati si possono migliorare ulteriormente le prestazioni della memoria associativa riducendo il numero di soluzioni spurie: sono casi di stati stabili della memoria che non corrispondono a nessun esempio memorizzato. Visto che usualmente queste

soluzioni spurie tendono ad avere bacini di attrazione più piccoli di quelli degli esempi memorizzati si possono ridurre genericamente tutti i bacini di attrazione sapendo che in questo modo colpirà per prime le soluzioni spurie. In pratica si fa partire la rete da uno stato casuale \vec{S} e la si lascia evolvere fino a che si stabilizza in uno stato $\vec{\eta}$, a questo punto si aggiornano le sinapsi con un termine di “dimenticanza”

$$w'_{ij} = w_{ij} - \frac{\varepsilon}{n} \eta_i \eta_j$$

che riduce la stabilità dello stato $\vec{\eta}$. Un modo più mirato di procedere è quello di mettere la memoria in uno stato iniziale che corrisponda ad uno degli stati memorizzati $\vec{\xi}_\mu$ corrotto dal rumore, si lascia poi evolvere la memoria che converge ad uno stato $\vec{\eta}$, a questo punto si aggiornano le sinapsi con

$$w'_{ij} = w_{ij} - \frac{\varepsilon}{n} (\eta_i \eta_j - \xi_{i\mu} \xi_{j\mu})$$

e in questo modo si aggiornano le sinapsi solo se la rete si è stabilizzata in uno stato $\vec{\eta} \neq \vec{\xi}_\mu$ e si tendono a “dimenticare” solo le soluzioni spurie. Questa dimenticanza selettiva migliora molto le prestazioni della memoria soprattutto in presenza di esempi con forte correlazione.

★ ★ ★

Infine menzioniamo che per memorizzare esempi con correlazioni c’è un modo di procedere completamente diverso: si possono preventivamente trasformare gli esempi in modo che gli esempi trasformati soddisfino le ipotesi di fattorizzazione della distribuzione (33) e poi usare questi eventi trasformati per memorizzarli nella memoria associativa; chi fosse interessato può trovare maggiori dettagli in [3].

11.3.1 Efficienza della memoria associativa

Analizziamo ora la richiesta di componenti totalmente casuali per gli esempi da memorizzare ξ_ν che abbiamo introdotto per riuscire a calcolare la probabilità di errore P_I per un singolo bit (dalla quale discende che le correlazioni fra i vari bit degli esempi sono tutte nulle). Se da un lato questa richiesta può apparire limitante dall’altro fa sì che la quantità di informazione contenuta negli esempi sia massima (vedi appendice A.5) e precisamente m esempi di questo tipo conterranno $n m$ bit di informazione (dimostratelo per esercizio). In altre parole così si presentano dati con il massimo contenuto di informazione, per esempio il risultato di un algoritmo di compressione come l’mp3. Questo vuol dire che la nostra analisi si applica al caso di una memoria che memorizza il massimo contenuto di informazione. Per cui per usarla sarà doppiamente utile farla precedere da una compressione dei dati: si risparmia nella dimensione della memoria e si ottiene un caso trattabile

analiticamente. Per rendersi conto, almeno qualitativamente, dell'efficienza della memoria associativa ci possiamo chiedere quanti bit di informazione ci sono necessari per memorizzare gli esempi. Se analizziamo i pesi definiti dalla regola di Hebb estesa (28) vediamo che, dato che

$$-m \leq nw_{ij} \leq m$$

e che i pesi possono assumere al più $m+1$ valori diversi, per memorizzare gli n^2 pesi avrò bisogno di $n^2 \lceil \log_2 m + 1 \rceil$ bit di informazione. A questi vanno sommati n bit per memorizzare lo stato dei neuroni. In totale dunque per costruire la memoria associativa avrò bisogno di

$$n + n^2 \lceil \log_2 m + 1 \rceil$$

bit di informazione. Se esprimiamo il numero di esempi come $m = \alpha n$ (e potremo raggiungere al più la capacità massima $\alpha \leq 0.138$), si vede che per memorizzare αn^2 bit ne devo usare $n + n^2 \lceil \log_2 \alpha n + 1 \rceil$ e il numero di bit necessari per memorizzarne uno risulta

$$\frac{n + n^2 \lceil \log_2 \alpha n + 1 \rceil}{\alpha n^2} = \frac{1}{\alpha} \left(\frac{1}{n} + \lceil \log_2 \alpha n + 1 \rceil \right) \simeq \frac{1}{\alpha} \log_2 \alpha n$$

rapporto che non appare tanto malvagio dato che la memoria associativa, oltre a memorizzare gli esempi, può anche recuperarli da informazioni parziali o rumorose, per cui è chiaro che deve avere una certa “ridondanza”. In altre parole il numero di bit aggiuntivi che necessita non è inefficienza ma è usato per garantire maggiori capacità.

11.3.2 Suggerimenti per simulare una memoria associativa

Terminiamo questa parte iniziale sulle memorie associative proponendo una ricetta pratica per programmarne una al computer:

1. dati m esempi di n bit ciascuno si generano i pesi secondo la regola di Hebb (28);
2. se si vuole si possono “migliorare” i pesi con i metodi illustrati nel paragrafo 11.3;
3. si mette nei neuroni un esempio parziale e/o rumoroso che si vuole recuperare dalla memoria;
4. si genera un numero casuale fra 1 e n per scegliere il neurone da aggiornare;
5. si aggiorna il neurone secondo la regola (26);
6. finché ci sono neuroni da aggiornare si ritorna al punto 4, altrimenti è finito, dai neuroni si legge l'esempio ritrovato.

Alcuni commenti riguardo all'implementazione pratica delle memorie associative:

- ricordarsi di caricare un numero di esempi $m < 0.138 n$, altrimenti la rete non funziona;
- come osservato nel paragrafo 11.3 conviene tenere nulli i termini diagonali della matrice dei pesi $w_{ii} = 0$;
- gli esempi da memorizzare dovrebbero avere probabilità del singolo bit $1/2$ e i bit dovrebbero essere indipendenti. Dato che esempi di questo tipo non si presentano molto interessanti da vedere, in subordine si potrebbero cercare degli esempi nei quali perlomeno la correlazione fra le varie coordinate sia nulla sia all'interno del singolo esempio che fra esempi diversi (anche questa scelta esclude le immagini). Se invece si vogliono memorizzare immagini allora la forte correlazione presente negli esempi peggiora di molto il funzionamento della memoria e bisogna ridurre drasticamente α , in pratica una memoria di $30 \times 30 = 900$ pixel riesce a memorizzare circa 5 immagini ($\alpha \simeq 0.005$). In alternativa si può scrivere un programma per il “miglioramento” dei pesi ma questo richiede un po' più di tempo;
- per lo stop, verso la fine, si può fare ogni tanto un loop su tutti i neuroni per vedere se ci sono ancora neuroni che devono cambiare stato, in altre parole se è già stato raggiunto uno stato stabile²⁷;
- alla fine volendo si può far partire la memoria da stati iniziali casuali e contare quante volte si stabilizza su uno degli stati memorizzati: in questo modo si fa una misura sperimentale della dimensione dei bacini di attrazione dei singoli esempi.

11.4 Uso della funzione Energia nelle memorie associative

Definiamo una funzione Energia (il nome apparirà chiaro tra poco) per una memoria associativa come una funzione $H(\vec{S})$ dello stato della memoria \vec{S} che, per brevità, annoteremo come $H_{\vec{S}}$

$$H_{\vec{S}} = -\frac{1}{2} \sum_{ij} S_i S_j w_{ij} \quad (36)$$

²⁷un metodo molto più efficace dal punto di vista della simulazione è quello di mantenere una lista dei neuroni da aggiornare ad ogni passo e fermarsi quando questa lista è vuota. Questo si fa facilmente calcolando una volta all'inizio tutti gli input dei neuroni con $\vec{l} := W\vec{S}$ e confrontando i segni di \vec{l} con quelli di \vec{S} , i segni diversi indicano i neuroni da aggiornare. Scelto un neurone a caso fra questi, chiamiamolo k , si cambia il segno dello stato di questo neurone $S'_k \rightarrow -S_k$ il che equivale a fare $\vec{S}' \rightarrow \vec{S} - 2S_k \vec{v}_k$ (dove \vec{v}_k rappresenta il versore che ha coordinate tutte 0 meno la k -esima che vale 1). A questo punto la nuova lista dei neuroni da aggiornare si calcola facilmente con $\vec{l}' = W\vec{S}' = W\vec{S} - 2S_k W\vec{v}_k = \vec{l} - 2S_k \vec{w}_k$.

e studiamo come evolve $H_{\vec{S}}$ in ogni aggiornamento dei neuroni

$$\Delta H = H_{\vec{S}'} - H_{\vec{S}} .$$

Consideriamo il sistema in uno stato \vec{S} e supponiamo che si aggiorni il neurone k . Per prima cosa osserviamo che se $S'_k = S_k$ ovviamente $\Delta H = 0$ per cui basta studiare il caso $S'_k = -S_k$. Considerando solo i termini di $H_{\vec{S}}$ nei quali appare esplicitamente S_k troviamo

$$\Delta H = H_{\vec{S}'} - H_{\vec{S}} = -\frac{1}{2} \sum_j S'_k S_j (w_{kj} + w_{jk}) + \frac{1}{2} \sum_j S_k S_j (w_{kj} + w_{jk}) =$$

ponendo $S'_k = -S_k$ troviamo

$$= \sum_j S_k S_j (w_{kj} + w_{jk}) = S_k \sum_j S_j (w_{kj} + w_{jk})$$



Figura 50: Aleksandr Lyapunov (1857–1918)

funzione limitata essendo una somma finita di quantità limitate²⁸) il sistema resta fisso in quello stato. In altre parole i minimi (assoluti e relativi) della funzione $H_{\vec{S}}$ sono stati stabili della rete. Si dice in questo caso che la funzione $H_{\vec{S}}$ è una *funzione di Lyapunov* (figura 50) del sistema che è una funzione importante introdotta nella teoria delle equazioni differenziali ordinarie, e perciò nei sistemi dinamici, che permette di semplificare lo

e supponendo $w_{kj} = w_{jk}$ (relazione che per esempio è soddisfatta dalla (28)) questa diventa

$$\Delta H = 2S_k \sum_j w_{kj} S_j \quad (37)$$

che è sicuramente minore di zero dato che avevamo supposto che fosse

$$S'_k = \operatorname{sgn} \left(\sum_j w_{kj} S_j \right) = -S_k.$$

Dunque abbiamo provato che, purché $w_{kj} = w_{jk}$, ad ogni passo di aggiornamento o nessun neurone cambia stato e $\Delta H = 0$ o, se un neurone cambia stato, si ha $\Delta H < 0$ il che prova che ad ogni passo di evoluzione $H_{\vec{S}}$ non può che diminuire e, di conseguenza, quando $H_{\vec{S}}$ raggiunge un valore minimo ($H_{\vec{S}}$ è una

²⁸osserviamo che in forma matriciale possiamo scrivere $H_{\vec{S}} = -\frac{1}{2} S^T W S$ e $S^T S = n$ e $S^T W S$ è limitato fra gli autovalori minimi e massimi di W , che è simmetrica.

studio della stabilità. Si trova che se è definibile una funzione di Lyapunov essa fornisce una condizione sufficiente per la stabilità del sistema, inoltre si possono studiare le soluzioni stabili del sistema senza far intervenire la dinamica ma semplicemente studiando i minimi della funzione di Lyapunov. Questo tipo di funzioni si incontra spesso e prende diversi nomi a seconda del campo al quale si applica: Hamiltoniana per i sistemi fisici, funzione costo per i sistemi di ottimizzazione combinatoria e funzione di fitness in biologia dell’evoluzione.

In generale non sempre si può definire una funzione di Lyapunov, ed anzi non è nemmeno una condizione necessaria per la stabilità (cioè esistono sistemi che hanno stati stabili pur non avendo una funzione di Lyapunov) però quando si può definirla rende più semplice lo studio dei sistemi dinamici. Per esempio, nel nostro caso, non occorrerà indagare la dinamica della memoria associativa ma basterà studiare i minimi di $H_{\vec{S}}$ che coincidono proprio con i punti stabili studiati in precedenza (29). Questo non trascurabile vantaggio lo abbiamo ottenuto pagando un certo prezzo: i pesi simmetrici sono chiaramente un’ipotesi in totale disaccordo con l’evidenza biologica: basti pensare a come sono fatte le sinapsi: non c’è nessuna ragione per cui i due pesi sinaptici che collegano due neuroni distinti debbano essere uguali.

Per capire dove possano essere i minimi di $H_{\vec{S}}$ riprendiamo la (36) e sostituiamo il valore dei pesi dalla regola di Hebb (28)

$$\begin{aligned} H_{\vec{S}} &= -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n S_i S_j w_{ij} = -\frac{1}{2n} \sum_{i=1}^n \sum_{j=1}^n \sum_{\nu=1}^m S_i S_j \xi_{i\nu} \xi_{j\nu} = \\ &= -\frac{1}{2n} \sum_{\nu=1}^m \left(\sum_{i=1}^n S_i \xi_{i\nu} \right) \left(\sum_{j=1}^n S_j \xi_{j\nu} \right) = -\frac{1}{2n} \sum_{\nu=1}^m (\vec{S} \cdot \vec{\xi}_{\nu})^2 \end{aligned}$$

notiamo che è una somma di termini positivi tutti compresi fra 0 e n^2 . Se riandiamo al caso di un solo esempio (27) vediamo che il minimo di questa funzione, $-\frac{n}{2}$, si raggiunge solamente quando $\vec{S} = \pm \vec{\xi}$ riconfermando il caso studiato inizialmente e riprodotto in figura 44 che assume ora una nuova luce vista come il grafico della funzione $H_{\vec{S}}$.

Per più esempi memorizzati la cosa è più complicata ma per $\vec{S} = \pm \vec{\xi}_{\nu}$ almeno uno dei termini della somma di $H_{\vec{S}}$ vale n^2 (mentre tutti gli altri, per le proprietà della distribuzione degli esempi, hanno valore più piccolo (sapreste calcolarlo ?)) indicando che, verosimilmente, quello potrebbe essere un punto di minimo per $H_{\vec{S}}$. Dunque la regola di Hebb, applicata alla memoria associativa, fa sì che gli esempi memorizzati siano minimi per $H_{\vec{S}}$.

Possiamo anche ragionare al rovescio e, sapendo di avere a che fare con un sistema che ha una funzione di Lyapunov, partire costruendo una funzione $H_{\vec{S}}$ con minimi negli esempi che vogliamo memorizzare. Una scelta ovvia è proprio l’ultima formula appena ricavata; in questo caso svolgendo i passaggi dall’ultimo al primo troviamo i valori dei pesi w_{ij} che realizzano questa funzione e che, ovviamente, sono i pesi dati dalla regola di Hebb (28).

12 Meccanica statistica e reti neurali

Alla fine del 1800, sull'onda dell'interesse per i motori a combustione interna, la meccanica statistica ha vissuto il suo periodo d'oro. Ricordiamo qui alcuni concetti fondamentali che poi applicheremo alle memorie associative.

La meccanica statistica considera sistemi fisici formati da *moltissimi* sistemi elementari (e.g. 10^{23} atomi o molecole) in interazione reciproca. Più che il comportamento dei singoli elementi (non avrebbe molto senso conoscere 10^{23} posizioni e velocità) si cerca di descrivere le caratteristiche macroscopiche del sistema nel suo complesso (per esempio pressione, volume, magnetizzazione etc.) dato che queste sono le quantità misurabili facilmente.

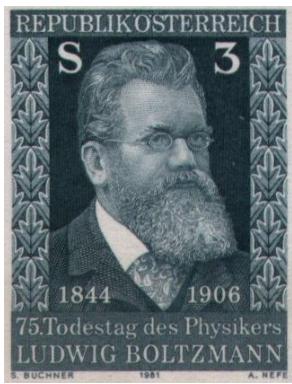


Figura 51: Ludwig Boltzmann (1844–1906)

Consideriamo sistemi in *equilibrio termico* come per esempio una bottiglia di gas in una stanza ad una certa temperatura assoluta $T > 0$. Se misuriamo, per esempio, la pressione sulle pareti, e disponiamo di strumenti abbastanza sensibili, osserviamo che la pressione varia nel tempo, di quantità molto piccole, a causa degli urti delle molecole con la parete. Questi urti sarebbero teoricamente prevedibili sulle basi delle equazioni del moto, e di conseguenza lo sarebbero le fluttuazioni della pressione, ma risulta più facile considerarle come *casuali*, introducendo la *probabilità* e trattarle *statisticamente*: questo approccio fa parte del concetto stesso di temperatura.

Detto \vec{S} un particolare stato microscopico del sistema, che in generale per $T > 0$ varia nel tempo, chiameremo $P_{\vec{S}}$ la probabilità di trovare il sistema in quel preciso stato. Data A una quantità che ha un particolare valore $A(\vec{S})$ quando il sistema si trova nello stato \vec{S} (ed è dunque soggetta a fluttuazioni nel tempo), calcoliamo il suo valore di aspettazione $E[A] := \langle A \rangle$ come

$$\langle A \rangle = \sum_{\vec{S}} P_{\vec{S}} A(\vec{S})$$

questa quantità, chiamata *media termodinamica*, fornisce il valore macroscopico di A che effettivamente si osserva in sistemi sufficientemente grandi ed è *indipendente dal tempo* per sistemi in equilibrio termico (viceversa se A dipende dal tempo non siamo in equilibrio). In questo modo possiamo calcolare le quantità osservabili (come $\langle A \rangle$) non dalle equazioni del moto dei singoli elementi ma, su basi statistiche, dalla probabilità $P_{\vec{S}}$, purché, lo ripetiamo, il sistema possa considerarsi *in equilibrio*.

Ludwig Boltzmann (figura 51), partendo da considerazioni molto generali sulla meccanica microscopica, ha dimostrato che, per un sistema in equilibrio termico, la probabilità $P_{\vec{S}}$ di trovarsi in uno stato \vec{S} è *indipendente*

dal tempo e dipende solo dall'energia totale dello stato $H_{\vec{S}}$ e risulta

$$P_{\vec{S}} \propto e^{-\frac{H_{\vec{S}}}{k_B T}}$$

dove $k_B = 1.38 \cdot 10^{-16} \text{ erg/K}$ è la costante di Boltzmann e T la temperatura assoluta; notiamo che lo stato più probabile è quello che ha energia totale $H_{\vec{S}}$ minima e come T regola la pendenza dell'esponenziale. Notiamo anche come, diversamente che per la dinamica standard, lo stato del sistema al tempo t non ha alcuna influenza sugli stati del sistema a t successivi. Per normalizzare correttamente le probabilità ci manda una costante e definiamo la, a prima vista innocua, *funzione di partizione* Z

$$Z = \sum_{\vec{S}} e^{-\frac{H_{\vec{S}}}{k_B T}}$$

con la quale possiamo finalmente scrivere la *distribuzione di Boltzmann* (o di Boltzmann–Gibbs)

$$P_{\vec{S}} = \frac{1}{Z} e^{-\frac{H_{\vec{S}}}{k_B T}} \quad (38)$$

che risulta ora normalizzata, $\sum_{\vec{S}} P_{\vec{S}} = 1$ e calcolare le medie termodinamiche

$$\langle A \rangle = \sum_{\vec{S}} P_{\vec{S}} A(\vec{S}) = \frac{1}{Z} \sum_{\vec{S}} e^{-\frac{H_{\vec{S}}}{k_B T}} A(\vec{S})$$

che predicono accuratamente le quantità misurabili purché il sistema sia in equilibrio e sia *ergodico* cioè possa accedere a tutti gli stati \vec{S} sui quali viene fatta la media termodinamica.

Dato che ci concentreremo sulle reti neurali la temperatura T diventa un parametro senza un preciso significato fisico per cui decidiamo di fissarne la scala in modo che scompaia la costante di Boltzmann cioè $k_B = 1$ e eliminiamo la costante da tutti i conti che seguono.

12.1 Il modello di Ising

Come esempio fisico, che poi applicheremo alle memorie associative, consideriamo un sistema di **Ising** in figura 52, cioè un reticolo di atomi dotati di momento magnetico (simile a un piccolo magnete) ed in particolare degli atomi con spin $\frac{1}{2}$ per i quali, si sa, il momento magnetico può essere orientato in due sole direzioni, chiamiamole su e giù e le indicheremo con $S = \pm 1$. In analogia al caso classico l'energia di un atomo nel campo magnetico h vale $-h S$. Questo modello, pur nella sua elementarità (trascura tutte le interazioni degli atomi esclusa quella magnetica!), è intensamente studiato dal 1920 perché fornisce un modello soddisfacente delle proprietà magnetiche della materia, inoltre il caso bidimensionale è l'unico caso di una transizione di fase che può essere trattata matematicamente in modo esatto.

Per studiare un modello di questo tipo occorre definire le *interazioni* (classicamente: le forze) e la *dinamica*. Cominciamo dalle interazioni calcolando il campo magnetico h_i che si trova sull' i -esimo atomo del reticolo

$$h_i = \sum_j w_{ij} S_j + \theta_i$$

dove la prima somma da il campo magnetico generato dai momenti magnetici degli altri atomi e w_{ij} è un fattore che tiene conto della interazione fra gli atomi S_i e S_j (distanza, tipo di interazione, etc.). Ovviamente in un sistema fisico le interazioni sono simmetriche per cui $w_{ij} = w_{ji}$. Il secondo termine, θ_i , rappresenta un eventuale campo magnetico esterno. Dato che l'energia dell' i -esimo atomo vale $-h_i S_i$ l'energia totale del reticolo in un certo stato \vec{S} vale

$$H_{\vec{S}} = - \sum_i h_i S_i = - \frac{1}{2} \sum_{ij} w_{ij} S_i S_j - \sum_i \theta_i S_i \quad (39)$$

dove il fattore $\frac{1}{2}$ tiene conto che nella \sum_{ij} le interazioni fra S_i e S_j vengono contate due volte. Osserviamo che è formalmente identica alla (36) introdotta nel paragrafo 11.4 purché si riprenda la vecchia definizione dei neuroni (1) rendendo di nuovo esplicita la soglia θ_i per ogni neurone per cui la regola di aggiornamento si scrive $S'_i = \text{sgn}(\sum_j w_{ij} S_j + \theta_i)$ e la funzione di Lyapunov (36) della memoria associativa è formalmente identica all'energia del reticolo.

Proviamo ad applicare i risultati della meccanica statistica a questo caso per esempio per calcolare una quantità osservabile come il valor medio dello spin dell'atomo k , cioè

$$\langle S_k \rangle = \sum_{\vec{S}} P_{\vec{S}} S_k(\vec{S}) = \frac{1}{Z} \sum_{\vec{S}} e^{-\frac{H_{\vec{S}}}{T}} S_k(\vec{S})$$

che è interessante in quanto ci dice se c'è una magnetizzazione media del sistema²⁹. La funzione di partizione è in questo caso la somma sui 2^n possibili stati dei neuroni della rete

$$Z = \sum_{\vec{S}} e^{\frac{1}{T} (\frac{1}{2} \sum_{ij} w_{ij} S_i S_j + \sum_i \theta_i S_i)}$$

dunque a prima vista potrebbe sembrare che per valutare $\langle S_k \rangle$ occorre calcolare per ben due volte una somma $\sum_{\vec{S}}$ su tutti i 2^n possibili stati dei neuroni

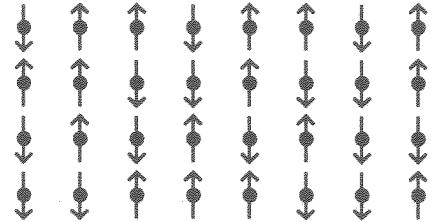


Figura 52: Schematizzazione di un sistema di Ising.

²⁹più precisamente è un *parametro d'ordine* della transizione di fase fra lo stato non magnetizzato, $\langle S_k \rangle = 0$, e quello magnetizzato, $\langle S_k \rangle \neq 0$.

della rete. In realtà osserviamo che conoscendo l'espressione analitica della funzione di partizione Z possiamo calcolare

$$\frac{\partial Z}{\partial \theta_k} = -\frac{1}{T} \sum_{\vec{S}} e^{-\frac{H_{\vec{S}}}{T}} \frac{\partial H_{\vec{S}}}{\partial \theta_k} = \frac{1}{T} \sum_{\vec{S}} S_k e^{-\frac{H_{\vec{S}}}{T}}$$

dunque possiamo scrivere

$$\langle S_k \rangle = \frac{T}{Z} \frac{\partial Z}{\partial \theta_k} = T \frac{\partial}{\partial \theta_k} \log Z$$

e se conosciamo l'espressione analitica della funzione di partizione Z potremo ricavare le quantità osservabili dai valori delle sue derivate.

Introducendo la funzione *energia libera* (di Helmholtz) che, dal punto di vista termodinamico, è l'energia a disposizione per fare lavoro,

$$F = -T \log Z$$

possiamo anche scrivere

$$\langle S_k \rangle = -\frac{\partial F}{\partial \theta_k}$$

Allo stesso modo si calcola facilmente la covarianza fra due neuroni:

$$\langle S_k S_l \rangle = -\frac{\partial F}{\partial w_{kl}}$$

che esprime la potenza del metodo che abbiamo introdotto: conoscere l'espressione analitica della funzione di partizione, o dell'energia libera, da la conoscenza pressoché totale sul sistema, dato che quasi tutte le quantità relative al sistema si possono calcolare a partire da queste con una semplice derivata (per esercizio calcolate $\langle H \rangle = T^2 \frac{\partial \log Z}{\partial T}$). Purtroppo il calcolo dell'espressione analitica della funzione di partizione quasi sempre è praticamente impossibile.

Ricordiamo la relazione fra potenziali termodinamici

$$U = F + T\mathcal{S}$$

dove \mathcal{S} è l'entropia termodinamica (attenzione a non confonderla con il vettore che rappresenta lo stato \vec{S}) e U l'energia totale del sistema che identifichiamo con $\langle H \rangle$. Come ulteriore esempio illustrativo calcoliamo la quantità

$$\langle H \rangle - F = \sum_{\vec{S}} P_{\vec{S}} H_{\vec{S}} + T \log Z = \sum_{\vec{S}} P_{\vec{S}} (H_{\vec{S}} + T \log Z) =$$

dove abbiamo usato la relazione di normalizzazione $\sum_{\vec{S}} P_{\vec{S}} = 1$

$$= -T \sum_{\vec{S}} P_{\vec{S}} \left(-\frac{H_{\vec{S}}}{T} - \log Z \right) = -T \sum_{\vec{S}} P_{\vec{S}} \left(\log e^{-\frac{H_{\vec{S}}}{T}} - \log Z \right) =$$

$$= -T \sum_{\vec{S}} P_{\vec{S}} \log \frac{e^{-\frac{H_{\vec{S}}}{T}}}{Z} = -T \sum_{\vec{S}} P_{\vec{S}} \log P_{\vec{S}}$$

che, confrontata con la relazione termodinamica ci permette di dedurre che l'entropia è

$$S = \sum_{\vec{S}} P_{\vec{S}} \log \frac{1}{P_{\vec{S}}}$$

che coincide (vedi appendice A.5) con la definizione del valor medio di quantità di *informazione* necessaria per specificare uno degli stati (a parte un fattore costante per la conversione dei logaritmi da base 2 a base e). Nel caso che il sistema che stiamo esaminando si trovi in W stati, tutti equiprobabili, si ha $P_{\vec{S}} = \frac{1}{W}$ e si trova allora facilmente

$$S = \log W$$

e dunque $e^S = W$. Spesso si generalizza questa relazione dando ad e^S il significato di numero di stati occupati dal sistema (o meglio: di volume di spazio delle fasi accessibile).

Concludiamo con un'altra relazione di uso frequente

$$e^{-\frac{F}{T}} = e^{\log Z} = Z = \sum_{\vec{S}} e^{-\frac{H_{\vec{S}}}{T}}$$

e ancora se ne deriva

$$\frac{e^{-\frac{F}{T}}}{Z} = 1 = \sum_{\vec{S}} P_{\vec{S}}$$

e anche questa si generalizza dando ad $\frac{e^{-\frac{F'}{T}}}{Z}$ il significato di probabilità di trovare il sistema in uno degli stati che ha energia libera F' . Infine da quanto visto poco sopra abbiamo

$$S = \frac{\langle H \rangle - F}{T} = T \frac{\partial \log Z}{\partial T} + \log Z = \frac{\partial T \log Z}{\partial T} = -\frac{\partial F}{\partial T}$$

e

$$T = \frac{\partial U}{\partial S} .$$

12.2 Semplici esempi fisici

Mettiamo alla prova questo approccio su un semplicissimo sistema formato da *un solo* atomo e iniziamo calcolandone il valore medio dello spin

$$\langle S \rangle = \sum_{S=-1}^1 P_S S = \dots = \frac{e^{\frac{h}{T}} - e^{-\frac{h}{T}}}{e^{\frac{h}{T}} + e^{-\frac{h}{T}}} = \tanh\left(\frac{h}{T}\right) \quad (40)$$

che in sostanza è riportata in figura 55 con la differenza che nella figura la tangente iperbolica è rinormalizzata fra 0 e 1.

Se pensiamo ad una sostanza composta da n atomi senza interazioni fra di loro ($w_{ij} = 0$) in pratica avremo un sistema composto da n dei singoli atomi appena descritti. Per questa ragione il momento magnetico complessivo del materiale sarà $M = n\langle S \rangle = n \tanh\left(\frac{h}{T}\right)$. Queste sostanze sono dette paramagnetiche (alluminio, aria etc.) e se studiamo la suscettività magnetica χ ossia come varia la magnetizzazione rispetto al campo magnetico (che può essere solo esterno) si ha

$$\chi = \frac{\partial M}{\partial h} = n \frac{\partial}{\partial h} \tanh\left(\frac{h}{T}\right) = \frac{n}{T} \left[1 - \tanh^2\left(\frac{h}{T}\right) \right]$$

e in sostanza si trova che per $h = 0$

$$\chi \propto \frac{1}{T}$$

la cosiddetta legge di (Pierre) Curie.

Esaminiamo infine un altro modello: quello di un ferromagnete, qui ci sono interazioni fra gli spin degli atomi e sono tutte positive, cioè $w_{ij} > 0$. Si sa che per i materiali di questo tipo, per esempio il ferro, esiste una transizione di fase ad una temperatura detta di Curie, per il ferro essa è di 768 °C; ben al di sotto della temperatura di fusione di 1.538 °C. Al di sotto della temperatura di Curie il ferro mostra una magnetizzazione spontanea e questa scompare se si supera questa temperatura. Vediamo come con il nostro modello possiamo riprodurre questo comportamento, almeno dal punto di vista qualitativo.

In particolare proviamo un modello estremamente semplice nel quale l'interazione sia uguale per tutti gli atomi (ignorando, per esempio, l'effetto della distanza)

$$w_{ij} = \frac{J}{n}$$

dove J è una costante. Anche in questo caso il valore medio di uno spin risulta ovviamente la tangente iperbolica del campo magnetico locale

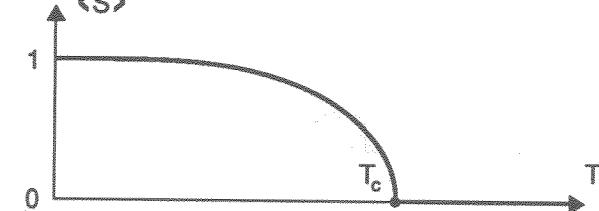


Figura 53: Magnetizzazione del ferro e transizione di fase alla temperatura critica T_C , il ramo della curva con magnetizzazione negativa non è disegnato.

$$\langle S_i \rangle = \tanh\left(\frac{h_i}{T}\right) = \tanh\left[\frac{1}{T} \left(\sum_j w_{ij} S_j + \theta_i \right)\right] \quad (41)$$

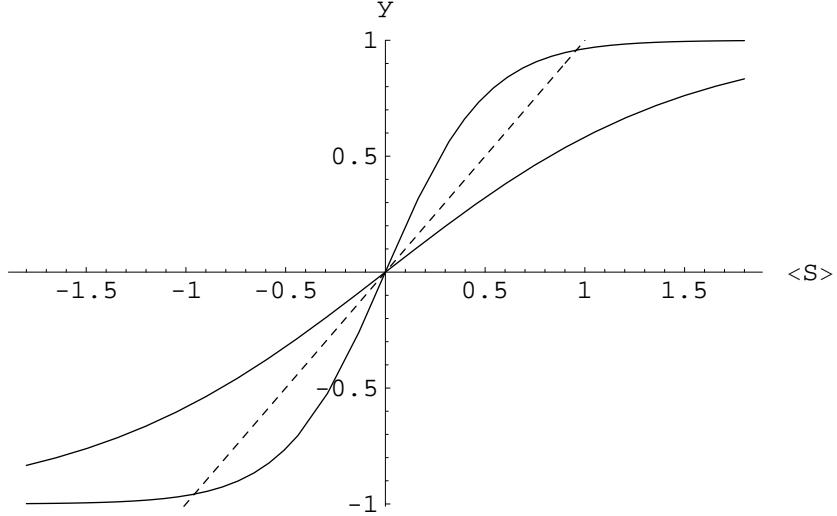


Figura 54: Soluzione grafica della (42) ad una temperatura maggiore ed una minore della temperatura di Curie; la linea tratteggiata è quella di equazione $y = \langle S \rangle$ e le sue intersezioni con $y = \tanh\left(\frac{J}{T}\langle S \rangle\right)$ danno le soluzioni.

che però non è risolubile se non altro perché dipende dalle variabili stocastiche S_j .

Per poter procedere dobbiamo usare l'*approssimazione di campo medio* (Mean Field Approximation) che approssima il valore istantaneo del campo magnetico h_j con il suo valore medio e cioè

$$h_i \simeq \langle h_i \rangle = \sum_j w_{ij} \langle S_j \rangle + \theta_i$$

che risulta esatta nel limite termodinamico $n \rightarrow \infty$ e con interazioni a raggio infinito (come nel nostro caso), ma solo approssimata per n finito oppure nel modello di Ising classico nel quale un atomo interagisce solo con i primi vicini. Con questa approssimazione e con il valore delle interazioni troviamo

$$\langle S_i \rangle = \tanh \left[\frac{1}{T} \left(\sum_j w_{ij} \langle S_j \rangle + \theta_i \right) \right] = \tanh \left[\frac{1}{T} \left(\frac{J}{n} \sum_j \langle S_j \rangle + \theta_i \right) \right] .$$

Vista la simmetria delle interazioni imposte e visto che il modello si immagina di dimensioni infinite è logico aspettarsi che i valori medi degli spin di tutti gli atomi siano uguali, cioè $\langle S_j \rangle = \langle S \rangle$. Con queste ipotesi, e con campo magnetico esterno nullo, troviamo

$$\langle S \rangle = \tanh \left(\frac{J}{T} \langle S \rangle \right) \quad (42)$$

che in figura 54 viene risolta graficamente per trovare $\langle S \rangle$. Si vede che pur con questo modello molto semplice c'è una temperatura critica $T_C = J$, la temperatura di Curie, al di sopra della quale c'è un'unica soluzione $\langle S \rangle = 0$ che indica che il materiale non ha magnetizzazione spontanea: il moto disordinato della temperatura non permette agli atomi di allinearsi. Ad una temperatura inferiore a T_C , oltre alla soluzione $\langle S \rangle = 0$ (che risulta instabile), emergono due altre soluzioni con magnetizzazione positiva e negativa. Questo semplicissimo modello riesce a riprodurre, almeno qualitativamente, la transizione di fase della magnetizzazione³⁰ che è ben nota per i materiali ferromagnetici ed è riprodotta in figura 53.

12.3 Cenni alla dinamica stocastica

Ci siamo lasciati alle spalle la dinamica delle equazioni del moto sostituendole con la distribuzione di Boltzmann però ancora non siamo in grado di calcolare nulla di un sistema come quello di Ising dato che non sappiamo calcolare la funzione di partizione Z . Possiamo però, con una capriola logica, reintrodurre la dinamica, ma non quella classica, reale, che integra le equazioni del moto, ma una *dinamica fittizia* che sia in grado di riprodurre la distribuzione di Boltzmann del nostro sistema e che ci servirà per studiare il sistema e soprattutto per simularlo al calcolatore. Introduciamo cioè una *dinamica stocastica* (casuale) che preveda il valore futuro dello spin di un atomo S'_i in funzione del campo magnetico presente, della temperatura etc. (classicamente: le equazioni del moto).

Da quanto visto per il modello di Ising (41) ci aspettiamo che sull'orientamento dello spin di un atomo agiscano due spinte contrastanti: il campo magnetico locale h_i e l'agitazione termica, indicata dalla temperatura T . A basse temperature l'agitazione termica è piccola e lo spin di un atomo si orienterà secondo il campo magnetico locale, ad alta temperatura prevarrà l'agitazione termica che tende ad orientare casualmente lo spin in maniera tanto più forte maggiore è la temperatura. Cioè ci aspettiamo che a $T = 0$ valga $S'_i = \text{sgn}(h_i)$ ma che a $T > 0$ la dipendenza sia meno rigida e in particolare per $T \rightarrow \infty$ ci aspettiamo che l'effetto del campo magnetico sia diventato trascurabile dato che $\langle S_i \rangle = 0$.

Per descrivere matematicamente le fluttuazioni termiche Glauber, nel 1963, ha introdotto un modello stocastico per la dinamica, che prescrive che lo spin del i -esimo atomo, dopo l'aggiornamento stocastico, sia

$$S'_i = \begin{cases} 1 & \text{con probabilità } p(h_i) \\ -1 & \text{con probabilità } 1 - p(h_i) \end{cases}$$

³⁰È una transizione di fase di secondo ordine, continua in M , ma con suscettività magnetica $\chi = \frac{\partial M}{\partial h}$ infinita.

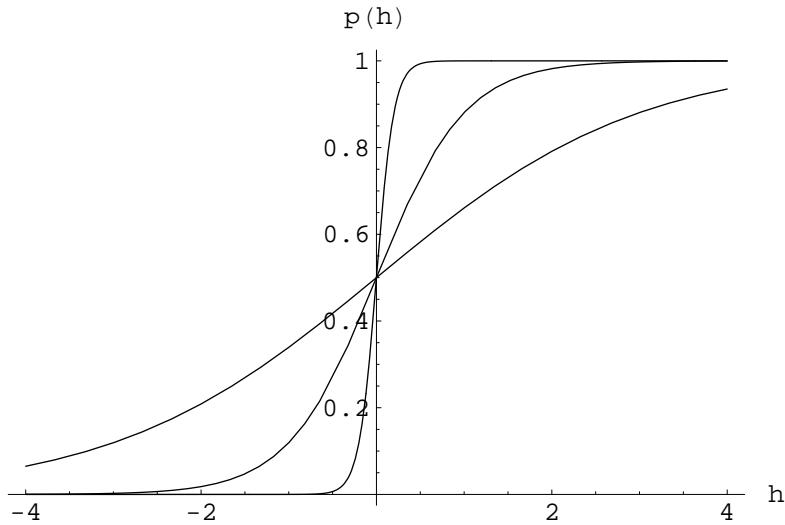


Figura 55: $p(h)$ in (43) plottata per tre diversi valori di temperatura, per $T \rightarrow 0$ la funzione tende a diventare una funzione a gradino.

dove

$$p(h_i) = \frac{1}{2} \left[1 + \tanh \left(\frac{h_i}{T} \right) \right] = \frac{1}{1 + e^{-\frac{2h_i}{T}}} \quad (43)$$

e inoltre prescrive che, durante l’evoluzione del sistema, sia nulla la probabilità che due atomi cambino stato contemporaneamente. Osserviamo che la funzione $p(h)$, rappresentata in figura 55, è tale che vale $1 - p(h) = p(-h)$. Con questa relazione si può sinteticamente esprimere la probabilità che S'_i assuma un certo valore

$$P(S'_i = \pm 1) = p(\pm h_i) = p(S'_i h_i) . \quad (44)$$

La dinamica stocastica di Glauber non è l’unica scelta possibile. In generale si può definire una dinamica stabilendo le probabilità di transizione $\mathcal{P}(\alpha \rightarrow \alpha')$ da uno stato α ad un nuovo stato α' . Fornire un insieme di probabilità di transizione è una specifica molto generale per un sistema dinamico e non è detto che conduca ad un caso di equilibrio, potremo trovare per esempio comportamenti ciclici, caotici etc. Per essere certi di arrivare al caso di un sistema in equilibrio (descritto nel nostro caso dalla distribuzione di Boltzmann) basta rispettare la condizione sufficiente – e spesso anche necessaria – di “bilancio dettagliato”

$$P_\alpha \mathcal{P}(\alpha \rightarrow \alpha') = P_{\alpha'} \mathcal{P}(\alpha' \rightarrow \alpha)$$

che, con la distribuzione di Boltzmann (38), da

$$\frac{\mathcal{P}(\alpha \rightarrow \alpha')}{\mathcal{P}(\alpha' \rightarrow \alpha)} = \frac{P_{\alpha'}}{P_\alpha} = e^{-\frac{H_{\alpha'} - H_\alpha}{T}} = e^{-\frac{\Delta H}{T}} \quad (45)$$

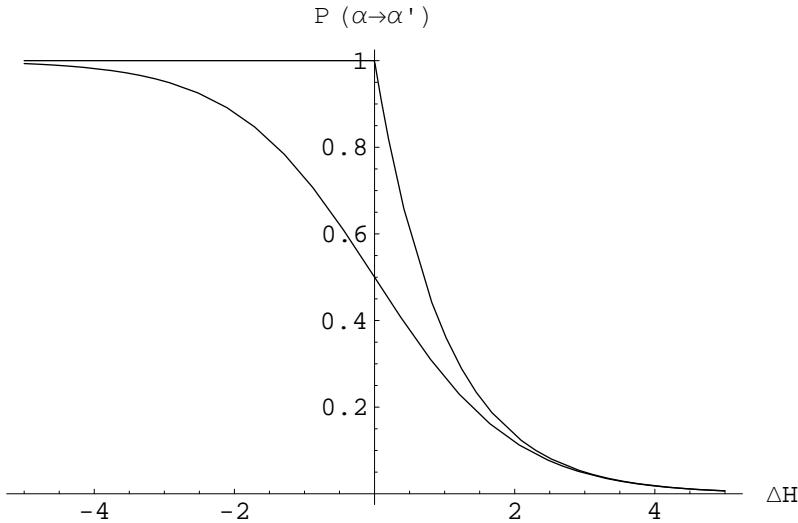


Figura 56: Grafico della probabilità di transizione $\mathcal{P}(\alpha \rightarrow \alpha')$ per le due regole di Glauber (46) e di Metropolis (47).

Verifichiamo se la dinamica stocastica di Glauber soddisfa questa condizione: sostituendo $-S_i$ ad S'_i nella (44) troviamo

$$\mathcal{P}(S_i \rightarrow S'_i = -S_i) = p(-S_i h_i)$$

e ricordando che la variazione di energia H quando $S_i \rightarrow -S_i$ vale $2h_i S_i$ (37) abbiamo, indicando questa variazione con ΔH ,

$$\mathcal{P}(S_i \rightarrow -S_i) = p\left(-\frac{\Delta H}{2}\right) = \frac{1}{2} \left[1 + \tanh\left(-\frac{\Delta H}{2T}\right) \right] = \frac{1}{1 + e^{\frac{\Delta H}{T}}} . \quad (46)$$

In questa forma è facile verificare che la regola di Glauber rispetta la condizione (45) dunque ora sappiamo che mantiene l'equilibrio. Con metodi più raffinati si può dimostrare anche che questa regola, applicata abbastanza a lungo, porta all'equilibrio, e dunque alla distribuzione di Boltzmann, a partire da *uno stato qualsiasi*.

La regola di Glauber è la più semplice da interpretare ma non è l'unica. Nel 1953 Metropolis, Rosenbluth, Rosenbluth e Teller (si, proprio quello della bomba H) inventano un metodo per simulare numericamente (ovverosia approssimare) le caratteristiche di un sistema a molti corpi

$$\mathcal{P}(\alpha \rightarrow \alpha') = \begin{cases} 1 & \text{se } \Delta H \leq 0 \\ e^{-\frac{\Delta H}{T}} & \text{altrimenti} \end{cases} \quad (47)$$

questa regola, producendo più transizioni, come si vede in figura 56, nelle simulazioni fa evolvere il sistema più rapidamente ed inoltre rispetta le condizioni per l'equilibrio (45) come è facile verificare. Un applicazione molto generale è l'algoritmo di “Simulated Annealing” descritto nell'appendice A.6.

12.4 Modello di Ising e memorie associative

Da quanto detto finora si vede che molte cose sono simili o anche formalmente identiche fra memorie associative e modelli di Ising. Riassumiamo le similitudini fin qui trovate con alcuni commenti:

	Memoria associativa	Modello di Ising
$S_i = \pm 1$	gli stati dei neuroni	gli spin del modello di Ising
w_{ij}	i pesi	le interazioni
$w_{ij} = w_{ji}$	indispensabile per definire una funzione di Lyapunov del sistema ma assai poco "biologica"	azione e reazione
θ_i	la soglia	il campo magnetico esterno
$\vec{w}_i \cdot \vec{S}$	l'input dell' i -esimo neurone	il campo magnetico dovuto agli altri spin
$H_{\vec{S}}$	la funzione di Lyapunov (36)	l'energia totale (39)
$S_i \rightarrow S'_i$	la regola di aggiornamento dei neuroni (26)	la dinamica stocastica di Glauber (46).

L'unica differenza sostanziale è proprio nella funzione di aggiornamento che è "deterministica" per i neuroni mentre è probabilistica per gli spin del modello di Ising. Ora proviamo che nel limite $T \rightarrow 0$ la dinamica stocastica di Glauber (46)

$$\mathcal{P}(S_i \rightarrow -S_i) = \frac{1}{2} \left[1 - \tanh \left(\frac{\Delta H}{2T} \right) \right]$$

diventa proprio quella deterministica delle reti neurali. Dato che per $T \rightarrow 0$ abbiamo $\tanh \left(\frac{\Delta H}{2T} \right) \rightarrow \text{sgn}(\Delta H)$ la dinamica stocastica diventa

$$\mathcal{P}(S_i \rightarrow -S_i) = \begin{cases} 1 & \text{se } \Delta H < 0 \\ 0 & \text{altrimenti} \end{cases}$$

che indica un aggiornamento deterministico che avviene se, e solo se, $\Delta H < 0$ e che possiamo formulare con

$$S'_i = \text{sgn}(S_i \Delta H) .$$

Ricordando che $\Delta H = 2h_i S_i$ si trova

$$S'_i = \text{sgn}(S_i^2 2h_i) = \text{sgn}(h_i) = \text{sgn}(\vec{w}_i \cdot \vec{S} + \theta_i)$$

che è proprio la regola di aggiornamento del singolo neurone adottata per la memoria associativa.

Invece che considerare i due sistemi coincidenti solo nel limite $T \rightarrow 0$ possiamo anche rovesciare la situazione e adottare una regola di aggiornamento probabilistica anche per le reti neurali incoraggiati anche dalla considerazione che nelle reti neurali biologiche c'è sempre del rumore (dovuto per esempio alla casualità del rilascio di neurotrasmettitori, alla velocità di trasmissione dei segnali nervosi e a fenomeni simili). Il grado di casualità o "rumore" è (rozzamente) rappresentato dal parametro T che per una rete neurale agisce come una "pseudo" temperatura non avendo per essa alcun significato fisico, inoltre abbiamo visto che abbassando T si può sempre ritornare alla regola di aggiornamento deterministica. Con questa generalizzazione vedremo che la memoria associativa funziona meglio perché riesce a sfuggire più facilmente a soluzioni spurie indesiderate che spesso hanno minimi relativi con bacini di attrazione molto piccoli e basta cambiare qualche bit per permettere alla rete di uscirvi.

Un notevole vantaggio che otteniamo con questa generalizzazione è che ora sappiamo per costruzione che se simuliamo questo sistema abbastanza a lungo da poterlo ritenere in equilibrio allora la probabilità di trovarlo in un certo stato \vec{S} sarà data dalla distribuzione di Boltzmann (38)

$$P_{\vec{S}} = \frac{1}{Z} e^{-\frac{H_{\vec{S}}}{T}}$$

e inoltre potremo applicare alle memorie associative tutti i risultati trovati per i modelli di Ising.

Per simulare al calcolatore una rete di questo tipo si procede così:

- si seleziona un neurone i a caso per un eventuale cambio di stato $S_i \rightarrow -S_i$,
- si calcola il relativo $\Delta H = 2h_i S_i$,
- si genera un numero a caso $0 \leq x \leq 1$ e si cambia stato al neurone S_i se (con la dinamica stocastica (47))

$$\Delta H < 0 \quad \text{oppure} \quad \Delta H > 0 \quad \text{e} \quad x \leq e^{-\frac{\Delta H}{T}}$$

e si procede così finché la rete non ha raggiunto l'equilibrio termodinamico.

12.5 Soluzioni del modello di Hopfield nel caso $\alpha \rightarrow 0$

Seguendo idealmente il percorso fatto per i sistemi di Ising consideriamo un sistema, il *modello di Hopfield* introdotto nel 1982, che a questo punto non chiamiamo più rete neurale, avendolo identificato totalmente con il modello di Ising. In questo modello i pesi w_{ij} sono dati dalla regola di Hebb (28) e, come visto nel paragrafo 11.3, hanno distribuzione Gaussiana centrata a 0 mentre la regola di aggiornamento è data dalla dinamica stocastica (47).

Uno dei primi tipi di soluzione che possiamo esaminare in questa nuova veste è quello di una rete che memorizza un solo esempio $\vec{\xi} = (1, 1, 1, \dots, 1)$; in questo caso la regola di Hebb per i pesi (28) ci da $w_{ij} = \frac{1}{n}$ e riconosciamo il caso da poco analizzato di un ferromagnete con $J = 1$. Dunque c'è completa identità fra il processo di allineamento degli spin di un ferromagnete con la direzione del campo e il processo di richiamo dell'unico esempio memorizzato in una memoria associativa³¹. Inoltre per quanto visto sui sistemi ferromagnetici sappiamo anche che c'è una transizione di fase a $T_c = J = 1$ e possiamo dedurre che la memoria funziona solo per $0 \leq T < 1$. Per questa ragione si chiama fase ferromagnetica quella tipica di una memoria associativa con un solo esempio memorizzato.

Esaminiamo ora il caso di una memoria nella quale si siano memorizzati con la regola di Hebb (28) un piccolo numero di esempi m ; più precisamente con questo si intende, visto che tutti i conti saranno fatti nel limite termodinamico $n \rightarrow \infty$, che

$$\lim_{n \rightarrow \infty} \alpha = 0 \quad \text{con} \quad \alpha = \frac{m}{n}$$

e, più precisamente, i risultati che seguono saranno validi fino a $m \simeq \log n$. Con m piccolo potremo pensare ad una piccola perturbazione del caso appena visto di un solo esempio e ci aspettiamo che la memoria continui a funzionare pressapoco nello stesso modo, vediamo come rendere questa affermazione più credibile.

Dato che il sistema è dotato di funzione di Lyapunov sappiamo che arriva sempre ad una condizione di equilibrio corrispondente ad un minimo di $H_{\vec{S}}$ (cioè non ci sono casi nei quali, per esempio, la rete si mette in uno stato di oscillazione permanente, questo almeno a $T = 0$). Per studiare questi minimi possiamo limitarci alle soluzioni dalle corrispondenti condizioni di stabilità che assumono la forma usuale

$$\langle S_i \rangle = \tanh\left(\frac{h_i}{T}\right) = \tanh\left[\frac{1}{T} \left(\sum_j w_{ij} S_j + \theta_i \right)\right] = \tanh\left[\frac{1}{T} \left(\frac{1}{n} \sum_j \sum_{\nu} \xi_{i\nu} \xi_{j\nu} S_j + \theta_i \right)\right]$$

e che, nell'approssimazione di campo medio e con campo magnetico esterno nullo, diventano

$$\langle S_i \rangle = \tanh\left(\frac{1}{nT} \sum_j \sum_{\nu} \xi_{i\nu} \xi_{j\nu} \langle S_j \rangle\right) .$$

Da questo punto non si può procedere senza delle ulteriori semplificazioni e introduciamo un “ansatz” (postulato) che afferma che, quando gli esempi memorizzati sono pochi, gli stati stabili sono proporzionali agli esempi

³¹le cose non cambiano se l'esempio memorizzato $\vec{\xi}$ ha segni qualsiasi dato che questo sistema è isomorfo ad un sistema $\tilde{S}_i = \xi_i S_i$ con il solo evento memorizzato $(1, 1, 1, \dots, 1)$.

memorizzati e cioè supponiamo che valga

$$\langle S_i \rangle = M \xi_{i\mu}$$

per uno qualsiasi degli eventi memorizzati e dove $0 \leq M \leq 1$ indica quanto “orientato” è l’i-esimo spin. Con questo assunto si trova

$$\langle S_i \rangle = M \xi_{i\mu} = \tanh \left(\frac{M}{nT} \sum_j \sum_\nu \xi_{i\nu} \xi_{j\nu} \xi_{j\mu} \right) = \tanh \left[\frac{M}{T} \left(\xi_{i\mu} + \frac{1}{n} \sum_j \sum_{\nu \neq \mu} \xi_{i\nu} \xi_{j\nu} \xi_{j\mu} \right) \right]$$

dove i passaggi sono quelli già incontrati nello studio della stabilità per più esempi memorizzati fatti nel capitolo 11. In quel caso avevamo trovato che la seconda somma non riesce a cancellare il segno di $\xi_{i\mu}$ (come appare nella (30)) se gli esempi memorizzati sono pochi, ma questo è proprio il nostro caso. Infatti dalla (34) si vede che per $\alpha \rightarrow 0$ il contributo della seconda somma è trascurabile con probabilità che tende a 1 e la nostra espressione si semplifica in

$$M \xi_{i\mu} = \tanh \left(\frac{M \xi_{i\mu}}{T} \right)$$

e dato che $\tanh(-x) = -\tanh(x)$ possiamo semplificare $\xi_{i\mu}$ e trovare alla fine che

$$M = \tanh \left(\frac{M}{T} \right)$$

cioè esattamente il caso ferromagnetico già studiato nella (42) per la quale abbiamo scoperto che esiste una transizione di fase alla memorizzazione, in questo caso a $T = 1$ e dunque lo stesso tipo di soluzione ferromagnetica vale per $1 \leq m \leq \log n$.

Ricordando la (40) possiamo scrivere per l’ansatz

$$\langle S_i \rangle = M \xi_{i\mu} = P(S_i = 1) - P(S_i = -1)$$

e dato che $P(S_i = x) = 1 - P(S_i = -x)$ possiamo riassumere le due equazioni per i due possibili valori di $\xi_{i\mu}$ con

$$M = 2P(S_i = \xi_{i\mu}) - 1$$

e dunque la probabilità che l’i-esimo spin abbia il valore corretto è

$$P(S_i = \xi_{i\mu}) = \frac{1}{2}(M + 1)$$

e dunque il numero di spin corretti, cioè coincidenti con quelli di un esempio memorizzato, vale

$$nP(S_i = \xi_{i\mu}) = \frac{n}{2}(M + 1)$$

e dato che la dipendenza di M dalla temperatura è quella rappresentata in figura 53 ne deduciamo che per tutte le temperature al disopra della

temperatura critica il numero di bit corretti è $n/2$ e che questo numero tende a n (o a 0, cioè $\langle S \rangle = -1$) quando $T \rightarrow 0$.

Vediamo ora come la generalizzazione di una memoria associativa al caso $T > 0$ (il che vuol dire essenzialmente con l'aggiunta di rumore casuale) possa migliorarne le caratteristiche. Abbiamo già visto che quando memorizziamo un esempio $\vec{\xi}_\mu$ anche l'esempio $-\vec{\xi}_\mu$ rappresenta un possibile stato stabile della rete; vedremo ora che non è l'unico esempio di soluzioni spurie, cioè soluzioni indesiderate. Supponiamo di aver memorizzato un certo numero di esempi con la regola di Hebb (28) e consideriamo il seguente esempio spurio

$$\xi_{is} = \text{sgn}(\pm \xi_{i\mu} \pm \xi_{i\eta} \pm \xi_{i\omega})$$

dove con il \pm indichiamo che tutte le 8 possibili combinazioni di segni sono possibili, noi per comodità studieremo quella con tutti i segni $+$ ³². Al solito studieremo la stabilità di ξ_{is} ottenendo per l'argomento di $\tanh()$ (e tralasciando le costanti)

$$\sum_j \sum_\nu \xi_{i\nu} \xi_{j\nu} \xi_{js} = \xi_{i\mu} \sum_j \xi_{j\mu} \xi_{js} + \xi_{i\eta} (\vec{\xi}_\eta \cdot \vec{\xi}_s) + \xi_{i\omega} (\vec{\xi}_\omega \cdot \vec{\xi}_s) + \sum_j \sum_{\nu \neq \mu, \eta, \omega} \xi_{i\nu} \xi_{j\nu} \xi_{js}$$

dove abbiamo come al solito isolato nella somma su ν i tre termini con $\nu = \mu, \eta, \omega$ che hanno dato origine ad altrettanti prodotti scalari fra i tre esempi $\vec{\xi}_\mu$, $\vec{\xi}_\eta$ e $\vec{\xi}_\omega$.

Dimostriamo ora che tutti e 3 prodotti scalari hanno valore di aspettazione $n/2$: ricordando la definizione di $\vec{\xi}_s$ è facile vedere che $\xi_{is} = -\xi_{i\mu}$ solamente quando $\xi_{i\eta} = \xi_{i\omega} = -\xi_{i\mu}$ e dato che abbiamo assunto che i singoli spin degli esempi debbano avere $p = 1/2$ di valere ± 1 e che sono tutti indipendenti fra di loro, questo avviene solo in 2 casi su 8; negli altri 6 casi $\xi_{is} = \xi_{i\mu}$. Dunque la probabilità p che uno dei termini nella somma dei prodotti scalari sia 1 vale $p = 3/4$ che inserita nella (31) prova che $E[\vec{\xi}_\mu \cdot \vec{\xi}_s] = n/2$ e dunque la solita somma diviene

$$\sum_j \sum_\nu \xi_{i\nu} \xi_{j\nu} \xi_{js} = \frac{n}{2} (\xi_{i\mu} + \xi_{i\eta} + \xi_{i\omega}) + \sum_j \sum_{\nu \neq \mu, \eta, \omega} \xi_{i\nu} \xi_{j\nu} \xi_{js} \simeq \frac{n}{2} \xi_{is} + \sum_j \sum_{\nu \neq \mu, \eta, \omega} \xi_{i\nu} \xi_{j\nu} \xi_{js}$$

dove la seconda somma contiene ora $n(m-3)$ termini. Dunque la condizione di stabilità per ξ_{is} si può scrivere

$$M\xi_{is} = \tanh \left[\frac{M}{T} \left(\frac{\xi_{is}}{2} + \frac{1}{n} \sum_j \sum_{\nu \neq \mu, \eta, \omega} \xi_{i\nu} \xi_{j\nu} \xi_{js} \right) \right]$$

³²sono anche possibili combinazioni di qualsiasi sottinsieme dispari maggiore di 3 di esempi memorizzati; i sottinsiemi pari non vanno bene perché potrebbero avere qualche componente uguale a 0.

che mostra che anche in questo caso l'esempio spurio $\vec{\xi}_s$ avrà una condizione di stabilità anche se leggermente diversa da quella degli esempi memorizzati per il fattore $1/2$ che fa sì che la seconda somma avrà vita più semplice nel cambiare il segno di ξ_{is} . Un analisi più dettagliata conferma che anche gli esempi spuri sono minimi dell'energia anche se meno profondi di quelli degli esempi veri. La cosa interessante è che si trova che anche per i minimi spuri c'è una transizione di fase ferromagnetica che però avviene a $T_c = 0.46$ al disopra della quale i minimi spuri non sono più punti di stabilità. Ne consegue che simulando una memoria associativa ad una temperatura $0.46 < T < 1$ saremo in una situazione nella quale i minimi spuri non sono più punti di stabilità ma lo sono ancora gli esempi memorizzati. In altre parole in queste condizioni, con l'aggiunta di rumore termico, la memoria associativa funziona meglio.

12.6 Il caso $\alpha \neq 0$

Il caso in cui nel limite termodinamico $n \rightarrow \infty$ il numero di esempi m è tale che α rimane costante è molto più complesso e noi non lo affronteremo limitandoci a qualche osservazione generale ed a una descrizione sommaria di qualche risultato.

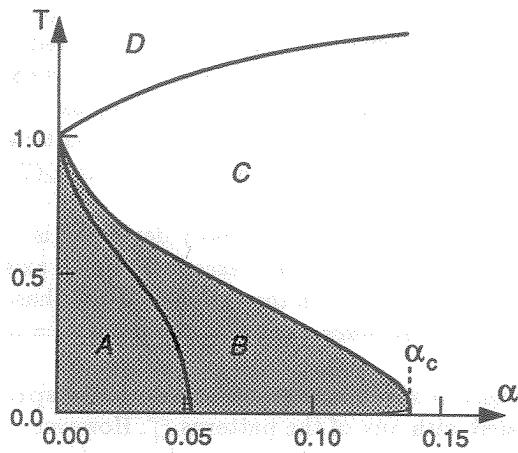


Figura 57: Diagramma di fase per il modello di Hopfield, sugli assi appaiono $\alpha = \frac{m}{n}$ e cioè il rapporto fra gli esempi memorizzati e il numero di neuroni e la temperatura T .

gli esempi memorizzati sono rappresentati con dei pallini neri e sono i minimi assoluti. Oltre il valore critico $\alpha_c = \frac{m}{n} = 0.138$ si passa a una situazione di tipo C nella quale gli esempi memorizzati non sono più minimi dell'energia e la memoria non è più in grado di richiamarli. Ad alta temperatura

La "fase ferromagnetica" della quale abbiamo appena parlato si ha quando gli esempi memorizzati sono pochi ed è indicata con la lettera A nelle figure 58 e 57 che contengono rispettivamente una rappresentazione qualitativa dei quattro tipi diversi di soluzioni ed un diagramma di fase con quattro aree distinte. Nelle aree ombreggiate, A e B di figura 57 il sistema funziona effettivamente come una memoria associativa e gli esempi memorizzati corrispondono a minimi di $H_{\vec{S}}$. Nella fase ferromagnetica A le zone "seghettate" della figura 58 rappresentano i minimi relativi dovuti alle soluzioni spuri mentre

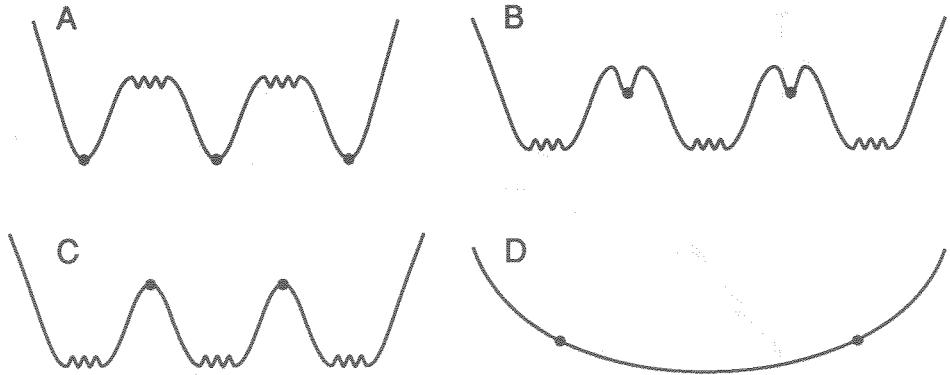


Figura 58: Descrizione qualitativa delle soluzioni nelle diverse fasi di figura 57; in ascissa è rappresentato lo stato della rete \vec{S} mentre in ordinata c'è l'energia $H_{\vec{S}}$, i pallini rappresentano gli esempi memorizzati $\vec{\xi}_\nu$.

$(T > 1)$ si entra nella fase D dove l'agitazione termica cancella ogni traccia di memoria e $\langle S_i \rangle = 0$.

La transizione di fase che separa le aree B e C è una transizione discontinua (detta di primo ordine) dato che $\langle S_i \rangle$ salta da un valore finito a 0. Solamente per $\alpha = 0$ e $T_c = 1$ la transizione di fase è continua in $\langle S_i \rangle$ (di secondo ordine) come abbiamo visto in figura 53. Mentre nell'area B gli esempi sono ritrovati con circa il 97% dei bit corretti quando si passa all'area C la percentuale di bit corretti per gli stati stabili è 50%, questo corrisponde ad una specie di *catastrofe cognitiva* che si ottiene quando si aggiunge anche un solo esempio a quelli già memorizzati superando il valore critico di $\frac{m}{n}$ per quella temperatura. Come si vede dalla figura 58 anche nell'area C ci sono molti stati stabili ma che risultano scorrelati dagli esempi memorizzati, questa zona è detta fase degli “[spin glass](#)” o vetri di spin per indicare il disordine strutturale che li caratterizza. In questa fase l'altezza dei picchi che separa i minimi cresce con n e nel limite termodinamico diventa infinita per cui il sistema non è più ergodico, cioè non visita tutti gli spazi disponibili dello spazio delle fasi per cui non può più dirsi in equilibrio termodinamico. Sono sistemi molto interessanti che sono tuttora molto studiati. Anche nella fase B sono presenti soluzioni, cioè minimi dell'energia, di tipo spin-glass corrispondenti ai minimi più profondi, mentre solo nella fase ferromagnetica A i minimi più profondi sono quelli degli esempi. Per approfondire questi argomenti guardate nei libri di Hertz, Krogh e Palmer [8] e Dotzenko [5].

12.7 La macchina di Boltzmann

Con le memorie associative abbiamo introdotto una matrice dei pesi W che ammette connessioni fra tutti i neuroni della rete e in questo senso è una generalizzazione di tutte le reti viste finora. I diversi tipi di reti sono distinte

solo dal fatto di avere la matrice dei pesi W strutturata in modo diverso: una colonna per il Perceptron, a blocchi per una rete feed-forward etc. etc. In pratica per vedere una memoria associativa come una rete feed-forward basta dividere mentalmente l'insieme dei neuroni in due parti: l'input e l'output. Questa divisione è virtuale ma a questo punto una memoria associativa può comportarsi da rete feed-forward: fissato l'input (cioè fissato lo stato di un sottinsieme dei neuroni) la si lascia evolvere; all'equilibrio leggeremo sui neuroni che chiamiamo output la risposta desiderata. In quest'ottica una memoria associativa non fa altro che memorizzare gli esempi di una rete feed-forward e i bacini di attrazione danno luogo alla generalizzazione.

Vediamo adesso come, anche per le memorie associative, ci possano essere esempi che non possono essere appresi: supponiamo di avere una rete con 3 neuroni e di voler memorizzare gli stati $(1, 1, 1)$, $(1, -1, -1)$, $(-1, 1, -1)$ e $(-1, -1, 1)$. Se per i pesi usiamo la regola di Hebb (28) troviamo facilmente che $w_{12} = w_{13} = w_{23} = 0$. Questo fa intuire che questi esempi non possono essere memorizzati nella rete (si può anche dimostrarlo). In realtà riguardano agli esempi e pensando ai primi due neuroni come input e al terzo come output riconosciamo un vecchio amico: l'XOR.

Per superare questa difficoltà, analogamente a quanto abbiamo fatto nelle reti feed-forward, possiamo aggiungere dei neuroni “nascosti”, cioè non visibili dall'esterno, che permettono di risolvere il problema e di memorizzare gli esempi desiderati. Nulla è cambiato dal punto di vista della memoria associativa: abbiamo solamente deciso che certi neuroni non sono visibili. Questo però crea un problema perché non sappiamo come determinare i pesi dei neuroni nascosti. Se per i neuroni “visibili” conosciamo lo stato da memorizzare per ogni esempio, per quelli nascosti ignoriamo lo stato; dunque avremo bisogno di una regola di apprendimento per calcolare i loro pesi. Ci viene in soccorso il fatto che possiamo calcolare la probabilità di trovare il sistema in un certo stato quando questi si trovi in equilibrio. Vediamo come.

Prima di iniziare ribadiamo che tutto quello che segue potrà essere adattato a qualsiasi tipo di rete introducendo una particolare matrice dei pesi W con l'unica condizione che i pesi siano simmetrici $w_{ij} = w_{ji}$.

Per procedere stabiliamo che i nostri neuroni sono divisi in due sottinsiemi: nel primo, indicato con α ci sono gli n neuroni visibili dall'esterno mentre nel secondo, indicato con β , ci sono i neuroni nascosti. Sappiamo calcolare l'energia associata con uno stato

$$H_{\vec{S}_{\alpha\beta}} = -\frac{1}{2} \sum_{ij} S_{i\alpha\beta} S_{j\alpha\beta} w_{ij}$$

e, per brevità, la indicheremo con $H_{\alpha\beta}$. La probabilità di trovare il sistema in uno stato $\vec{S}_{\alpha\beta}$ sarà

$$P_{\alpha\beta} = \frac{1}{Z} e^{-\frac{H_{\alpha\beta}}{T}}$$

con la quale possiamo facilmente calcolare la probabilità di osservare un particolare stato α dei neuroni visibili

$$P_\alpha = \sum_\beta P_{\alpha\beta} = \sum_\beta \frac{1}{Z} e^{-\frac{H_{\alpha\beta}}{T}} \quad (48)$$

dove con \sum_β indichiamo la somma su tutti i possibili stati dei neuroni nascosti β .

Ci restano da specificare gli esempi da memorizzare $\vec{\xi}_1, \dots, \vec{\xi}_m$: un modo diverso, e più adatto ad una rete probabilistica, consiste nel formulare la richiesta che la distribuzione degli stati per la rete in equilibrio termodinamico e a temperatura prossima allo 0, quando gli unici stati “permessi” sono quelli a energia minima, sia

$$Q_{\vec{S}_\alpha} = \begin{cases} \frac{1}{m} & \text{se } \vec{S}_\alpha = \vec{\xi}_\nu \quad \forall \nu = 1, \dots, m \\ 0 & \text{se } \vec{S}_\alpha \neq \vec{\xi}_\nu \end{cases}.$$

Se si trovano dei pesi che realizzino questa richiesta, mettendo la rete in uno stato iniziale qualsiasi e lasciandola evolvere fino all'equilibrio (sempre a bassa temperatura) sappiamo, per le proprietà della dinamica di Glauber, che andrà verso uno stato di equilibrio che ha probabilità diversa da 0 solo per gli esempi memorizzati. Dunque l'idea di formulare richieste alla memoria in forma di distribuzione di probabilità per gli stati di equilibrio è una generalizzazione dei metodi visti in precedenza. Inoltre abbiamo anche aggiunto la richiesta che i bacini di attrazione degli esempi $\vec{\xi}_\nu$ siano tutti uguali dato che una rete che realizza queste richieste avrà la proprietà che da qualsiasi stato iniziale parta andrà a finire, con uguale probabilità, in uno qualsiasi degli esempi memorizzati. Questo se non altro mostra che i bacini di attrazione hanno la stessa dimensione (in media $\frac{2^n}{m}$) anche se nulla garantisce che gli stati di un bacino invece di formare un insieme compatto non siano sparsi a macchie di leopardo nello spazio delle 2^n configurazioni.

A questo punto, stabiliti i nostri esempi, o meglio una distribuzione di probabilità desiderata per i neuroni visibili Q_α , possiamo calcolare la distanza di Kullback Leibler (appendice A.5) fra la distribuzione desiderata Q_α e quella realmente assunta dai nostri neuroni P_α

$$D_{KL}(Q_\alpha, P_\alpha) = \sum_\alpha Q_\alpha \log \frac{Q_\alpha}{P_\alpha} \geq 0$$

dove vale l'uguaglianza se, e solo se $Q_\alpha = P_\alpha$ per tutti gli stati α .

Possiamo ora cercare i pesi w_{ij} che minimizzano $D_{KL}(Q_\alpha, P_\alpha)$ e per farlo possiamo scendere lungo il gradiente della distanza per cui aggiorneremo i pesi con la variazione (e ricordando che i pesi w_{ij} compaiono solo in P_α)

$$\Delta w_{ij} = -\frac{\partial D_{KL}(Q_\alpha, P_\alpha)}{\partial w_{ij}} = -\sum_\alpha Q_\alpha \frac{P_\alpha}{Q_\alpha} \left(-\frac{Q_\alpha}{P_\alpha^2} \right) \frac{\partial P_\alpha}{\partial w_{ij}} = \sum_\alpha \frac{Q_\alpha}{P_\alpha} \frac{\partial P_\alpha}{\partial w_{ij}} .$$

Calcoliamo la derivata partendo dalla distribuzione marginale (48)

$$\frac{\partial P_\alpha}{\partial w_{ij}} = \sum_\beta \frac{\partial}{\partial w_{ij}} \frac{1}{Z} e^{-\frac{H_{\alpha\beta}}{T}} = \sum_\beta \left(\frac{1}{Z} \frac{\partial e^{-\frac{H_{\alpha\beta}}{T}}}{\partial w_{ij}} - e^{-\frac{H_{\alpha\beta}}{T}} \frac{1}{Z^2} \frac{\partial Z}{\partial w_{ij}} \right)$$

e calcoliamo le due derivate separatamente

$$\begin{aligned} \frac{\partial e^{-\frac{H_{\alpha\beta}}{T}}}{\partial w_{ij}} &= -\frac{1}{T} e^{-\frac{H_{\alpha\beta}}{T}} \frac{\partial H_{\alpha\beta}}{\partial w_{ij}} = \frac{1}{2T} e^{-\frac{H_{\alpha\beta}}{T}} S_{i\alpha\beta} S_{j\alpha\beta} \\ \frac{\partial Z}{\partial w_{ij}} &= \frac{\partial}{\partial w_{ij}} \sum_{\mu\nu} e^{-\frac{H_{\mu\nu}}{T}} = \frac{1}{2T} \sum_{\mu\nu} e^{-\frac{H_{\mu\nu}}{T}} S_{i\mu\nu} S_{j\mu\nu} \end{aligned}$$

che sostituiamo ottenendo

$$\frac{\partial P_\alpha}{\partial w_{ij}} = \frac{1}{2T} \sum_\beta \left(\frac{e^{-\frac{H_{\alpha\beta}}{T}}}{Z} S_{i\alpha\beta} S_{j\alpha\beta} - \frac{e^{-\frac{H_{\alpha\beta}}{T}}}{Z} \sum_{\mu\nu} \frac{e^{-\frac{H_{\mu\nu}}{T}}}{Z} S_{i\mu\nu} S_{j\mu\nu} \right)$$

nella quale possiamo riconoscere nella sommatoria del secondo termine il valor medio della covarianza

$$\langle S_i S_j \rangle = \frac{1}{Z} \sum_{\mu\nu} e^{-\frac{H_{\mu\nu}}{T}} S_{i\mu\nu} S_{j\mu\nu}$$

dunque

$$\frac{\partial P_\alpha}{\partial w_{ij}} = \frac{1}{2T} \left(\sum_\beta P_{\alpha\beta} S_{i\alpha\beta} S_{j\alpha\beta} - \sum_\beta \langle S_i S_j \rangle P_{\alpha\beta} \right) = \frac{1}{2T} \left(\sum_\beta P_{\alpha\beta} S_{i\alpha\beta} S_{j\alpha\beta} - P_\alpha \langle S_i S_j \rangle \right).$$

Sostituendo questo risultato nel calcolo di Δw_{ij} troviamo

$$\begin{aligned} \Delta w_{ij} &= \frac{1}{2T} \sum_\alpha \frac{Q_\alpha}{P_\alpha} \left(\sum_\beta P_{\alpha\beta} S_{i\alpha\beta} S_{j\alpha\beta} - P_\alpha \langle S_i S_j \rangle \right) \\ &= \frac{1}{2T} \left(\sum_{\alpha\beta} \frac{P_{\alpha\beta}}{P_\alpha} Q_\alpha S_{i\alpha\beta} S_{j\alpha\beta} - \sum_\alpha Q_\alpha \langle S_i S_j \rangle \right) \\ &= \frac{1}{2T} \left(\sum_{\alpha\beta} P_{\beta|\alpha} Q_\alpha S_{i\alpha\beta} S_{j\alpha\beta} - \langle S_i S_j \rangle \right) \end{aligned}$$

dove abbiamo usato la relazione

$$P_{\alpha\beta} = P_{\beta|\alpha} P_\alpha$$

che definisce la probabilità $P_{\alpha\beta}$ come il prodotto fra la probabilità di trovare il sistema in uno stato α per la probabilità condizionale $P_{\beta|\alpha}$ che le variabili

nascoste siano in uno stato β quando quelle “visibili” sono in uno stato α . Possiamo ora interpretare la quantità

$$\sum_{\alpha\beta} P_{\beta|\alpha} Q_\alpha S_{i\alpha\beta} S_{j\alpha\beta}$$

come il valor medio di $S_i S_j$ quando gli stati visibili stanno negli stati α con probabilità Q_α e mediate su tutti i possibili stati β compatibili con questi. In altre parole si tratta del valor medio di $S_i S_j$ quando però le variabili visibili assumono i valori permessi dalla distribuzione Q_α , cioè degli esempi; indicheremo questa media particolare con $\langle S_i S_j \rangle_{esempi}$.

In conclusione abbiamo trovato che per l'apprendimento la variazione dei pesi deve essere proporzionale a

$$\Delta w_{ij} = \frac{1}{2T} (\langle S_i S_j \rangle_{esempi} - \langle S_i S_j \rangle)$$

e ora esaminiamo alcune delle caratteristiche di questa regola di apprendimento:

- nonostante la derivazione sia stata complicata risulta semplice e intuitiva: il primo termine è in sostanza un apprendimento Hebbiano mentre il secondo tende a far “dimenticare” le correlazioni casuali del sistema, inoltre
- all'equilibrio, varrà

$$\langle S_i S_j \rangle_{esempi} = \langle S_i S_j \rangle$$

cioè le correlazioni medie dovranno essere uguali sia che il sistema sia lasciato “libero” sia che le variabili visibili siano obbligate ad assumere i valori degli esempi.

- Se applichiamo questa regola ad una rete senza unità nascoste e assumendo tutti gli esempi equiprobabili con probabilità $Q_\alpha = \frac{1}{m}$, troviamo

$$\Delta w_{ij} = \frac{1}{2T} \langle S_i S_j \rangle_{esempi} = \frac{1}{2nT} \sum_{\nu=1}^m \xi_{i\nu} \xi_{j\nu}$$

dove la media viene fatta solo sugli esempi dato che non si può mediare su variabili nascoste che non esistono e ritroviamo la buona vecchia regola di Hebb (28) per le memorie associative.

- infine è estremamente verosimile dal punto di vista biologico perché per aggiornare un peso (sinapsi) c’è bisogno solo dell’informazione *lo-*

cale³³ di quella sinapsi: le correlazioni dei neuroni collegati per suo tramite³⁴.

Il rovescio della medaglia è che l'apprendimento per queste reti è incredibilmente lento: se vogliamo simulare al calcolatore questa rete occorrono quattro loop annidati uno dentro l'altro:

- il ciclo più esterno su tutti i pesi per ognuno dei quali devo calcolare Δw_{ij} e aggiornare i pesi,
- per calcolare ogni aggiornamento Δw_{ij} devo calcolare $\langle S_i S_j \rangle_{esempi}$ e $\langle S_i S_j \rangle$ i quali a loro volta sono calcolati facendo le medie temporali sui valori delle correlazioni $\langle S_i S_j \rangle$, cioè si lascia il sistema evolvere e si calcola il valore medio di $S_i S_j$ su un grande numero di applicazioni della regola di aggiornamento (46). Notiamo che per valutare $\langle S_i S_j \rangle_{esempi}$ dovrò avere i neuroni visibili in *tutti* gli stati degli esempi con probabilità Q_α ;
- per ogni calcolo di una covarianza $\langle S_i S_j \rangle_{esempi}$ occorre portare il sistema in equilibrio ad una data temperatura che deve essere piuttosto bassa per permettere alla rete di funzionare come desiderato. Questo significa che si deve applicare la regola di aggiornamento un numero molto grande di volte per portare la rete in equilibrio. Infatti a bassa temperatura la rete fa molta difficoltà a uscire dai minimi relativi di H . Per questa ragione risulta in pratica molto più veloce simulare la rete a temperatura inizialmente alta che viene poi abbassata lentamente mantenendo sempre il sistema in equilibrio; in pratica si usa la tecnica del “Simulated Annealing” descritta nell'appendice A.6. Questo ci porta alla necessità di un quarto ciclo:
 - per ogni temperatura applicare la regola di aggiornamento un numero sufficiente di volte per far sì che la rete si trovi in equilibrio a quella temperatura.

e in conclusione si capisce che eseguire questa procedura al calcolatore possa essere estremamente lento.

Questo tipo di reti, proposte da Hinton e Sejnowski nel 1983 e chiamate macchine di Boltzmann, se da un lato hanno proprietà teoriche assai

³³Oltre a questo tipo di rete usano informazione locale il Perceptron e il neurone di Oja mentre, per esempio, l'algoritmo di back-propagation di una rete feed-forward, presentato nel capitolo 6, ha bisogno di informazione non locale per funzionare. Dalla (13) si vede facilmente che i neuroni dell'input hanno bisogno delle informazioni dei neuroni degli strati successivi.

³⁴Un'interpretazione più fantasiosa legge nella regola di apprendimento il ciclo sonno veglia dei sistemi biologici: durante la veglia arrivano gli stimoli sensoriali che “fissano” i neuroni visibili ai valori degli esempi memorizzati per cui viene calcolato il termine $\langle S_i S_j \rangle_{esempi}$. Durante il sonno mancano gli stimoli esterni e la rete, in attività “libera”, apprende con il termine $\langle S_i S_j \rangle$.

interessanti e verosimiglianza biologica, dall'altro sono incredibilmente lente nell'apprendimento. Nondimeno quando applicate ad esempi concreti si dimostrano essere nettamente superiori alle altre reti finora incontrate (per esempio le reti feed-forward).

12.8 Ancora la regola di Hebb

Ricordiamo che Hebb (vedi paragrafo 4.3 e figura 12) ha introdotto l'omonima regola basandosi su osservazioni biologiche ma vale la pena di ricordare quante volte l'abbiamo ritrovata nei modelli di reti neurali. È stata alla base dell'algoritmo del Perceptron (capitolo 5) riemergendo nelle reti feed-forward (paragrafo 6.2), l'abbiamo usata come punto di partenza nell'apprendimento non supervisionato (paragrafo 9.1) e nelle memorie associative (capitolo 11) per le quali risulta essere la matrice di covarianza degli esempi (paragrafo 11.3). Per le memorie associative produce un'interpretazione della funzione di Lyapunov associata e viene anche ritrovata partendo dalla funzione di Lyapunov (paragrafo 11.4) e infine salta fuori come regola di apprendimento per la macchina di Boltzmann (paragrafo 12.7): non può essere una coincidenza!

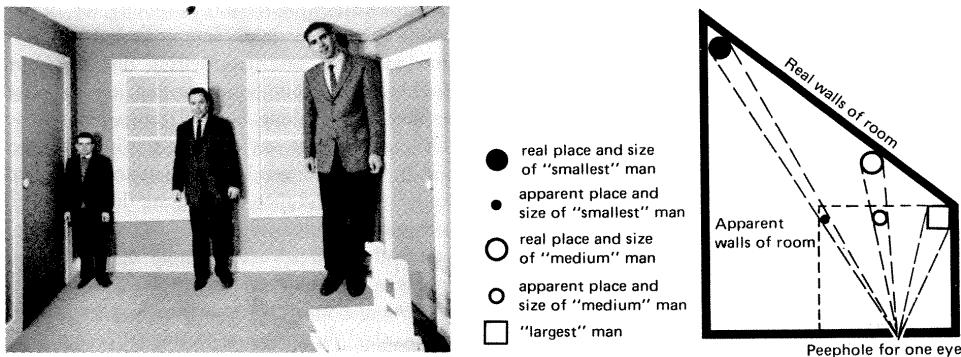


Figura 59: Informazione conflittuale dalla stanza di Ames: si tratta di una stanza deformata che però appare rettangolare da un particolare punto di vista. La prospettiva è dominante e le persone sembrano di dimensioni molto diverse pur avendo tutte la stessa altezza.

13 Una breve introduzione alla visione

Uno dei sistemi sensoriali più affascinanti e studiati negli organismi viventi è la visione che è un senso che, negli animali superiori, permette di costruire un immagine mentale tridimensionale del mondo circostante a partire da una sua proiezione bidimensionale sulla retina.

Nel caso dell'uomo è la maggior sorgente di informazione della quale disponiamo e, nonostante l'apparente semplicità con la quale ci sembra funzionare, si rivela un sistema altamente *efficiente, complesso ed adattato*, fin nei più minimi dettagli, all'ambiente che ci circonda.

Dopo una sommaria descrizione del sistema visivo umano accenneremo a qualche aspetto elementare di questo sistema per cercare di capire come si possa affrontarlo con un sistema neurale e, in particolare, parleremo dei problemi della correzione di uniformità e del rilevamento dei bordi di un'immagine (edge detection). Discuteremo poi brevemente di problemi *mal posti* (ill-posed problems) che è un concetto matematico indispensabile per affrontare i problemi della visione.

Il problema tipico della visione è ricostruire uno spazio tridimensionale a partire da una sua proiezione bidimensionale sulla retina. Dal semplice esempio in figura 60 si capisce che l'immagine sulla retina è compatibile con una infinità di oggetti nello spazio: questo è per definizione un problema mal posto. Questa caratteristica, unita al fatto che la visione non si accontenta di fornirci solo gli stimoli, ma fornisce sempre una loro interpretazione, è alla base di molte illusioni ottiche, come quella in figura 59.

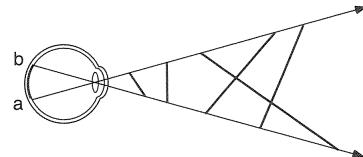


Figura 60: Problema inverso della retina: un'immagine retinica può derivare da infinite configurazioni nell'ambiente.

Lo studio della visione (naturale e artificiale) è ormai una scienza a se che affonda le sue radici nella *psicofisica* che dal 1850, con [Hermann von Helmholtz](#) (si, proprio il fisico), inizia ad interessarsi della misura quantitativa delle sensazioni e degli stimoli e ha risposto a domande del tipo: qual è la massima risoluzione angolare dell'occhio ? come si misura la persistenza dell'immagine nella retina ? qual è la risposta spettrale dei recettori di colore, i coni ? etc. Le note che seguono sono tratte dal Palmer [12].

13.1 Cenni alla visione umana

Con la parola visione implicitamente indichiamo una serie di processi cognitivi, di astrazione via via crescente, che possiamo sintetizzare nella lista

- trattamento dell'immagine;
- riconoscimento delle superfici;
- riconoscimento degli oggetti;
- categorizzazione.

noi ci concentriamo nella cosiddetta visione elementare (low level vision) che riguarda essenzialmente il trattamento delle immagini bidimensionali che si formano sulla retina tralasciando i passi successivi che si riferiscono allo spazio a tre dimensioni. Trascuriamo anche l'aspetto, assai studiato in passato, della percezione del colore dato che non è un elemento essenziale della visione, infatti, ci sono molti animali, specialmente notturni, che ne sono praticamente privi.

Lo studio della visione elementare è iniziato molto tempo fa ma fino agli anni del 1950 si è dovuto basare solo sulle defezioni visive derivanti da lesioni al cervello. Nel 1953 si sviluppano le prime tecniche per la registrazione *in vivo* dell'attività di neuroni visivi e le conoscenze aumentano molto rapidamente. Parallelamente, negli anni dal 1970, si sviluppa un approccio computazionale che mette alla prova le teorie sviluppate dalle osservazioni biologiche e cerca di riprodurre al calcolatore sistemi visivi ispirati ai sistemi biologici. Entrambi gli approcci sono stati fondamentali per l'avanzamento delle conoscenze sulla visione.

Per poter intuire, almeno in parte, la complessità della visione iniziamo con una descrizione sommaria delle sue parti principali seguendo idealmente il cammino dell'informazione dalla retina alla corteccia visiva come si vede in figura 61.

13.1.1 La retina

La retina è la parte dell'occhio sensibile alla luce ed è composta da 5 strati di cellule rappresentati in figura 64. I recettori sono in totale circa

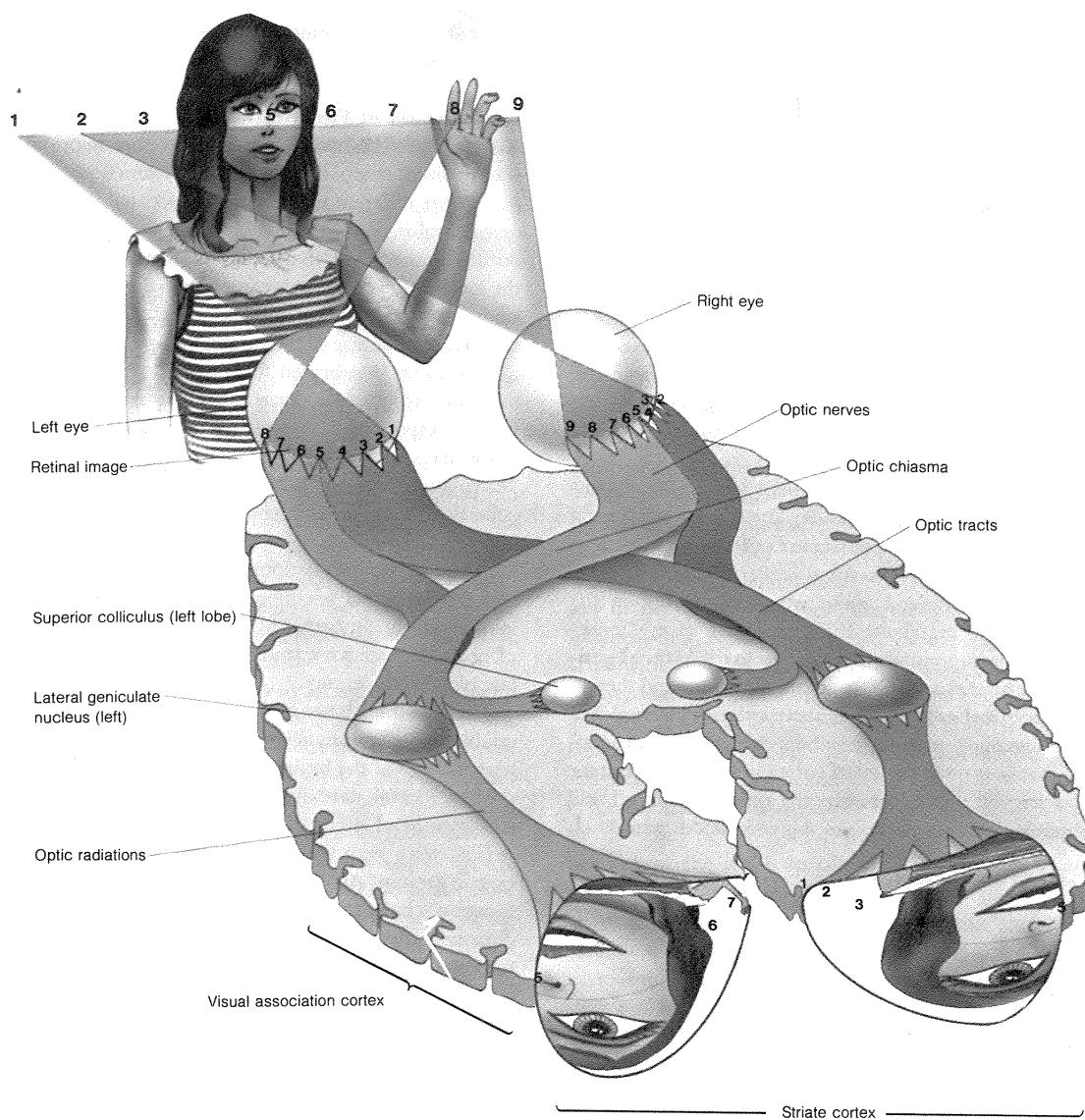


Figura 61: Principali elementi e connessioni del sistema visivo umano.

10^8 e non sono distribuiti uniformemente su tutta la retina ma sono molto più densi in un punto centrale, detto fovea, dove abbiamo la massima acuità visiva, vedi figura 62³⁵. I recettori sono di due tipi: i bastoncelli, più sensibili alla luce ma insensibili al colore, disposti essenzialmente fuori dalla fovea. I coni, sensibili al colore, sono di 3 tipi, sensibili a 3 lunghezze d'onda diverse; sono meno sensibili alla luce e sono concentrati vicino alla fovea.

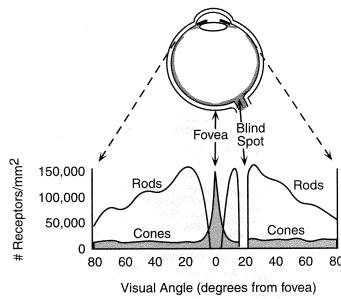


Figura 62: Distribuzione di coni e bastoncelli (cones & rods) nella retina: i coni sono concentrati nella fovea ed i bastoncelli all'esterno.

I recettori trasformano linearmente l'intensità luminosa in una differenza di potenziale che viene trasmessa agli strati di neuroni successivi. I primi quattro strati di neuroni non sono neuroni assimilabili a quelli di McCulloch e Pitts in quanto non sparano ma indicano con un potenziale continuo (graded potential) il loro stato. L'ultimo strato, quello dei neuroni gangliari, è fatto da neuroni che sparano e i loro assoni, lunghi alcuni centimetri, escono dall'occhio passando per il punto cieco³⁶, formano il nervo ottico e terminano nel nucleo genicolato laterale (Lateral Geniculate Nucleus o LGN). Il nervo ottico è formato da circa $1.2 \cdot 10^6$ assoni di neuroni gangliari.

Nel 1953 si è scoperta una tecnica che permette di monitorare l'attività di un neurone gangliare di un animale (gatto o macaco) cui vengono contemporaneamente mostrati stimoli visivi. Da questi studi si è scoperto che un neurone gangliare è connesso solamente ad un piccolo sottinsieme di recettori immediatamente sottostanti (vedi figura 64) ed ha una frequenza di sparo proporzionale alla quantità di luce che incide sui recettori cui è collegato. Più precisamente si è trovato che per un neurone gangliare esiste un'area circolare nella quale a maggior luce corrisponde maggior frequenza di sparo e di una corona circolare che la circonda che, se illuminata, fa diminuire la frequenza di sparo, vedi figura 63. In altre parole ogni neurone gangliare ha un area recettiva centro/periferia nella quale l'iluminazione produce effetti opposti e da origine alla famosa curva di risposta dell'area recettiva a forma di "cappello messicano" (mexican hat) riprodotta in figura 65. Con questo si intende che i recettori immediatamente sottostan-

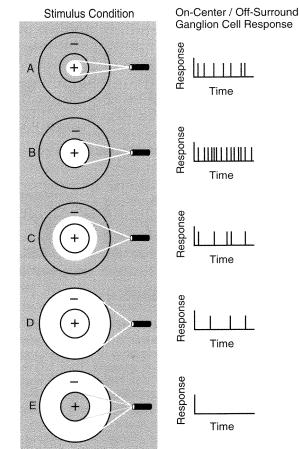


Figura 63: Risposta dei neuroni gangliari allo stimolo luminoso: in un area centrale la luce aumenta la frequenza di sparo, nella corona circolare la diminuisce. Illuminazione produce effetti opposti e da origine alla famosa curva di risposta dell'area recettiva a forma di "cappello messicano" (mexican hat) riprodotta in figura 65. Con questo si intende che i recettori immediatamente sottostan-

³⁵come termini di paragone una macchina fotografica da 33 Mpixel avrebbe circa 10^8 elementi sensibili (3 per colore) e la fovea, 184 in altra unità di misura, circa 10^4 dpi.

³⁶calamari e polipi non lo hanno.... vedi su [wikipedia](#)

ti un neurone gangliare sono collegati con sinapsi eccitatorie mentre quelli della corona circolare sono connessi con sinapsi inibitorie. L'intensità delle sinapsi varia, con la distanza radiale dal neurone gangliare, secondo una curva a forma di cappello messicano.

Questa curva (o la sua generalizzazione bidimensionale) viene anche detta *campo recettivo* dato che rappresenta la risposta che si ottiene dal neurone spostando uno stimolo puntiforme costante su un area centrata sul neurone in esame. Matematicamente si può pensare di stimolare solo il recettore x_i ottenendo dal neurone la risposta $x_i w_i$ dato che tutti gli altri recettori hanno input nullo; dunque il plot rappresenta, di fatto, la dipendenza dei pesi sinaptici w dalla posizione.

Le cellule intermedie fra i neuroni gangliari e i recettori, non sparando, sono più difficili da monitorare ma da quando si è riusciti a registrarne l'attività hanno mostrato, in sostanza, un funzionamento simile a quelle dei neuroni gangliari con una risposta centro/periferia che anche per esse ha la forma a cappello messicano.

Infine, per esercizio, ampliamo l'esempio già visto nell'appendice A.5 e valutiamo un limite superiore alla quantità di informazione che arriva sulla retina e che il sistema visivo deve “digerire”. Assumendo che i recettori della retina forniscano 10 immagini al secondo (il tempo di persistenza sulla retina è di circa 1/10 di secondo) e che ogni recettore dia 20 bit di informazione (circa 7 bit per ogni colore) ogni occhio produce, al più, 20 Gigabit al secondo di informazione.

Per contro si stima che solo una minima frazione dell'informazione che arriva sulla retina viene trasmessa lungo il nervo ottico indicando che una drastica riduzione di informazione viene fatta dalla retina stessa. Per confronto uno schermo di 1000×1000 pixel ognuno dei quali ha 24 bit di informazione (8 per ognuno dei tre colori fondamentali) ripetuto 10 volte al secondo produce, al più, 0.24 Gigabit al secondo di informazione.

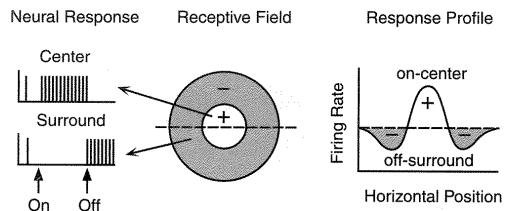


Figura 65: Campo recettivo e risposta agli stimoli delle cellule gangliari in funzione della distanza, una sezione ne mostra la tipica risposta a forma di cappello messicano.

13.1.2 Il nucleo genicolato laterale LGN

Come si vede in figura 61, gli assoni dei neuroni gangliari, usciti dall'occhio, si separano in due parti, ognuna corrispondente alla metà destra e sinistra della retina, e arrivano al nucleo genicolato laterale dopo l'incrocio nel chiasma ottico³⁷. Il nucleo genicolato laterale è formata da circa $1.5 \cdot 10^6$ neuroni

³⁷Solo qualche percento degli assoni va al collicolo superiore, legato al controllo dei movimenti oculari

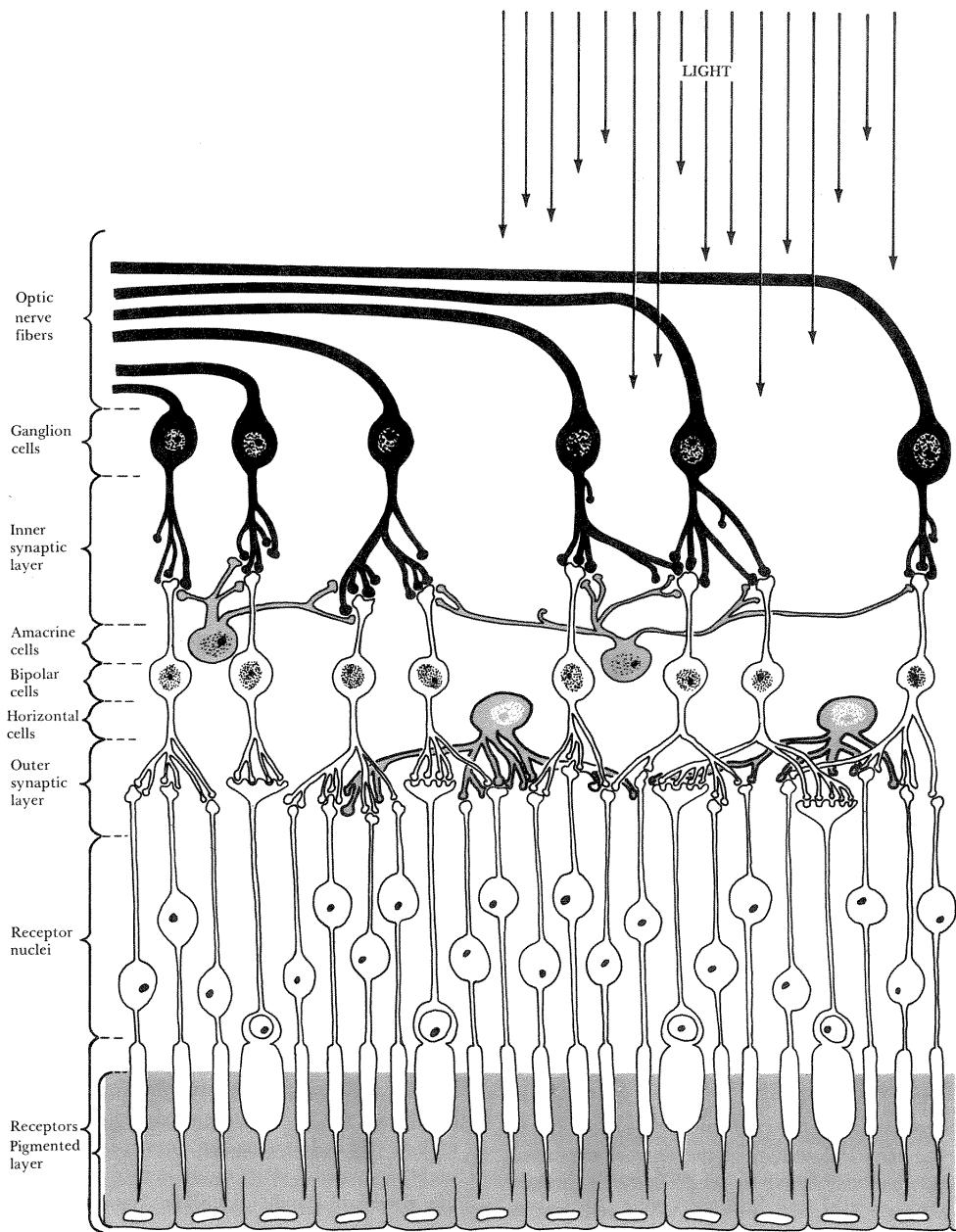


Figura 64: Sezione trasversale di retina umana. È formata da 5 tipi di neuroni, dal basso: recettori (bastoncelli e tre tipi di coni), cellule orizzontali, bipolari, amacrine e gangliari (da Lindsay & Norman, 1977).

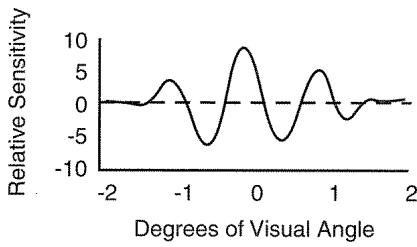


Figura 66: Campo recettivo di un neurone complesso con più lobi.

disposti in 6 lame connesse alternativamente all’occhio destro e sinistro. Esso non ha neuroni binoculari, cioè che ricevano informazioni da entrambi gli occhi, ma invece mantiene fedelmente la topologia delle connessioni in modo che a neuroni vicini nel nucleo genicolato laterale corrispondono recettori vicini sulla retina, la cosiddetta *mappa retinotopica*. Anche nel nucleo genicolato laterale si trova una mappa delle connessioni a cappello messicano ma usualmente di raggio maggiore e con inibizione più marcatà. Esaminando questa struttura in dettaglio si osserva una certa differenza fra gli strati: alcuni sono più sensibili a certe caratteristiche come colore e risoluzione spaziale, altri al contrasto e alla risoluzione temporale. Queste osservazioni sono alla base di una teoria che sostiene che nella visione ci sono trattamenti specializzati come colore, movimento, profondità spaziale, etc. che vengono portati avanti in parallelo, e senza contatti reciproci, fino all’analisi finale.

13.1.3 La corteccia striata

Come si vede in figura 61, gli assoni dei neuroni del nucleo genicolato laterale sono connessi, attraverso le radiazioni ottiche (optic radiations), alla corteccia striata che si trova nella zona della nostra nuca. La corteccia striata è un’area di circa 15 cm^2 , spessa solo 2 mm., formata da circa 200×10^6 neuroni ed è la sede, come indicato anche dal numero di neuroni, dei processi più complicati della visione. Nei primati è l’area corticale più grande deputata ad un unico scopo. Negli anni del 1950 Hubel e Wiesel, che riceveranno il premio Nobel per questi studi, riescono a registrare l’attività di singoli neuroni della corteccia visiva e iniziano lo studio di questa parte del cervello. Il procedimento è simile a quello usato per la retina: iniziata la registrazione dell’attività di un neurone si cercano gli stimoli visivi che lo eccitano. Questo studio si rivela assai meno facile di quello della retina e, quasi per caso³⁸, gli autori scoprono che alcuni neuroni della corteccia visiva reagiscono a linee e bordi nell’immagine. Col procedere dello studio trovano che le cellule della corteccia sono di tre tipi che chiamano semplici,

³⁸Dall’involontario scorrere di una linea molto marcata sullo schermo ottengono una forte risposta da un neurone che non sembrava reagire a nessuno stimolo.

complesse e ipercomplesse con riferimento agli stimoli che le eccitano. In estrema sintesi le cellule semplici reagiscono *linearmente* a bordi o barre nell'immagine; il loro campo recettivo ha dei lobi, come per i neuroni gangliari, ma in una direzione il campo recettivo è molto più allungato come mostrato in figura 67. Inoltre vi si trovano frequentemente neuroni con il campo recettivo con più lobi come mostrato in figura 66 ed esistono diversi tipi di cellule che reagiscono a bordi di dimensioni spaziali diverse.

Le cellule complesse, che sono la maggioranza, sono invece fortemente *non lineari* e reagiscono preferenzialmente al *moto* di linee e bordi ed anche in modo abbastanza indipendente dalla posizione dello stimolo nella retina, inoltre di solito hanno campi recettivi più grandi. Le cellule ipercomplesse reagiscono a bordi e linee terminati, cioè che nell'immagine si interrompono bruscamente.

Senza addentrarci nei dettagli gettiamo solo uno sguardo sull'architettura generale di quest'area del cervello sottolineando alcune delle sue caratteristiche più salienti.

- *Mappe retinotopiche*: nella corteccia a neuroni vicini corrispondono aree vicine nella retina ma le proprietà metriche sono distorte dato che per esempio i recettori della fovea occupano un area più grande di quelli delle parti laterali della retina. In un certo senso la fovea risulta “ingrandita”, vedi anche in figura 61 la faccia della ragazza.
- *Dominanza oculare*: le zone connesse ai due occhi sono nettamente separate ma finemente intrecciate: in figura 68 sono marcate in bianco e nero le zone della superficie della corteccia connesse rispettivamente all'uno e all'altro occhio.
- *Struttura colonnaire*: nel dettaglio la mappa retinotopica è costituita da aree più piccole, dette *ipercolonne*, che attraversano lo spessore della corteccia e che contengono, in zone vicine, neuroni sensibili a stimoli simili. Nell'esempio in figura 69, si trovano vicini, in un'unica ipercolonna, gruppi di neuroni sensibili a bordi di direzione via via diversa, si veda anche la figura 2.

Vista questa sofisticata organizzazione viene da chiedersi se queste strutture siano presenti sin dalla nascita. La risposta è: in parte; le strutture

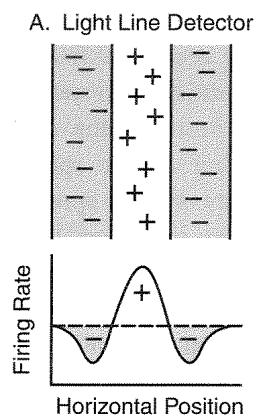


Figura 67: Campo recettivo di un neurone complesso sensibile ad una linea chiara e sua sezione trasversale.

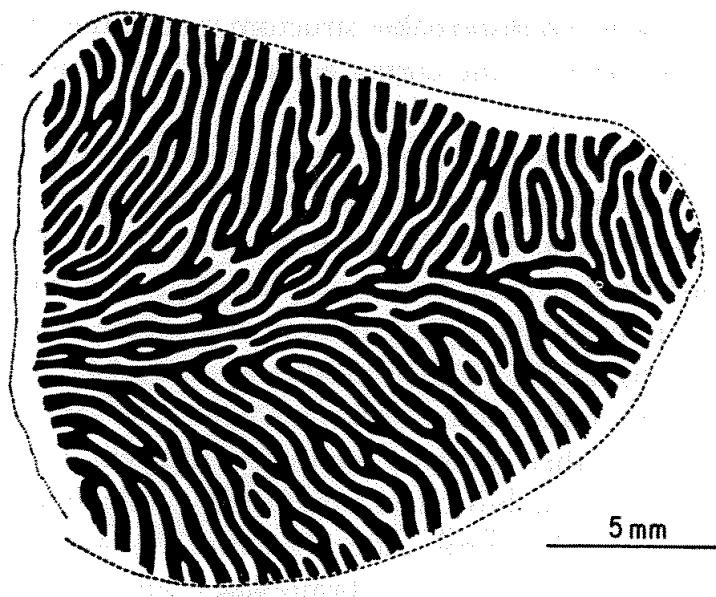


Figura 68: Sezione trasversale di corteccia visiva di macaco dove sono colorati in nero i neuroni provenienti da un solo occhio.

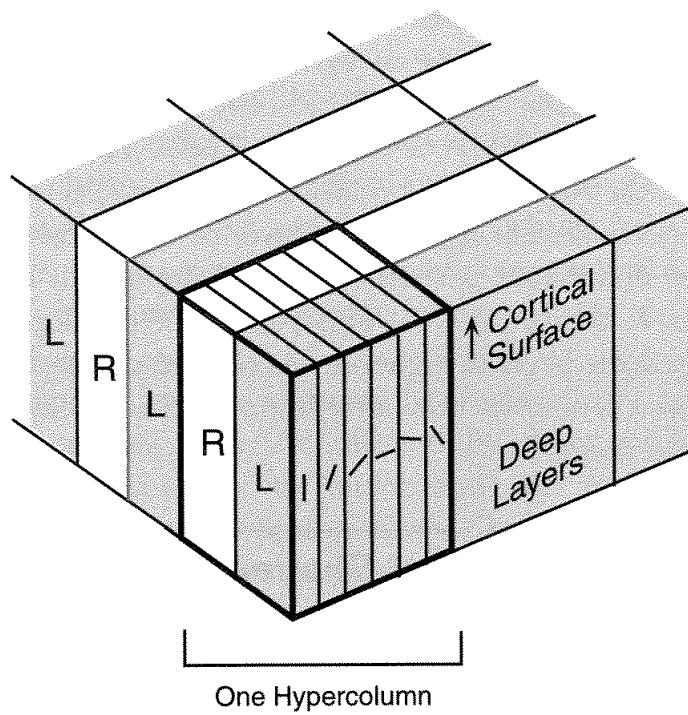


Figura 69: Organizzazione in ipercolonne nella corteccia striata: con L e R sono marcate le zone connesse ai due occhi; si notino zone dedicate al riconoscimento di linee di diversa direzione.

sono appena accennate alla nascita e si sviluppano fino a prendere la forma definitiva nel primo periodo di vita. In particolare c'è un periodo critico per lo sviluppo di ogni struttura (a iniziare da quelle più semplici) durante il quale la struttura si forma, guidata anche dagli stimoli visivi che riceve. Se, per qualsiasi ragione, in questo periodo le strutture non riescono a formarsi compiutamente (mancanza di stimoli, stimoli non rappresentativi di immagini naturali, etc.) la struttura non si aggiusta più in seguito e quella parte della visione è compromessa per sempre.

Tutta questa architettura appare complessa e articolata e ne conosciamo moltissimi dettagli ma in realtà, dopo 60 anni di ricerca, non sappiamo ancora *nulla* sulla sua interpretazione funzionale e cioè qual'è lo scopo e la funzione delle diverse parti rispetto al prodotto finale: la visione. Naturalmente ci sono delle ipotesi, e ne presentiamo subito una, lasciando quella computazionale di Marr per il paragrafo successivo.

13.2 La teoria della frequenza spaziale

È noto che una vasta classe di funzioni matematiche può essere sviluppata in serie di Fourier; per esempio un onda quadra $f(x)$ dispari di periodo 2π si può scrivere facilmente come una sovrapposizione di funzioni trigonometriche di frequenza diversa

$$\begin{aligned} f(x) &= \sum_{n=0}^{\infty} C_n \sin nx = \frac{4}{\pi} \sum_{n=0}^{\infty} \frac{\sin(2n+1)x}{2n+1} = \\ &= \frac{4}{\pi} \left(\sin x + \frac{1}{3} \sin 3x + \frac{1}{5} \sin 5x + \dots \right) \end{aligned} \quad (49)$$

vedi figura 70. Analogamente al caso della scomposizione di un vettore nello spazio reale

$$\vec{v} = \sum_{n=1}^3 v_n \vec{i}_n$$

i coefficienti C_n della somma (49) si possono pensare come le coordinate della funzione onda quadra in uno spazio delle funzioni a infinite dimensioni dove le funzioni trigonometriche $\sin nx$ giocano il ruolo dei versori. In questa ottica, così come un vettore è identificato univocamente dalle sue coordinate (v_1, v_2, v_3) , una funzione è identificata dai suoi (infiniti) coefficienti di Fourier che per l'onda quadra sono $\frac{4}{\pi}(1, 0, \frac{1}{3}, 0, \frac{1}{5}, 0, \dots)$.

Questo approccio si estende facilmente a funzioni definite su uno spazio bidimensionale $z = f(x, y)$ e si trova che anche queste funzioni si possono pensare come somma di funzioni sinusoidali con diverse frequenze e direzioni³⁹.

Si può applicare questa versione estesa del teorema di Fourier alle immagini (in bianco e nero) pensate come funzioni $z = f(x, y)$ e si trova che tutte le immagini possono essere viste come una somma di funzioni sinusoidali di diversa direzione e frequenza. Anche qui potremo pensare ai coefficienti di

³⁹ Per esempio $\sin(\alpha x + \beta y)$ è una sinusoide con le “creste” nella direzione $\alpha x + \beta y = \text{costante}$

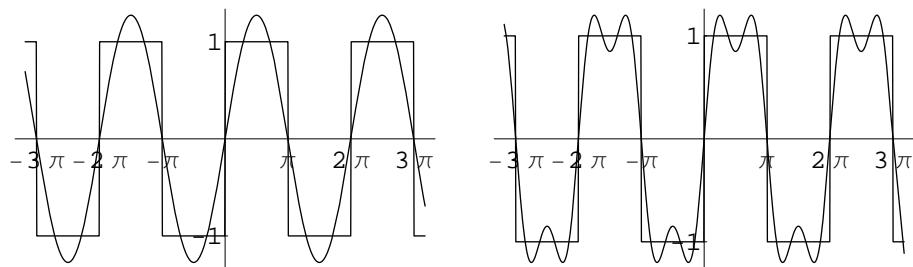


Figura 70: Onda quadra e sue approssimazioni con 1 e 2 termini di Fourier.

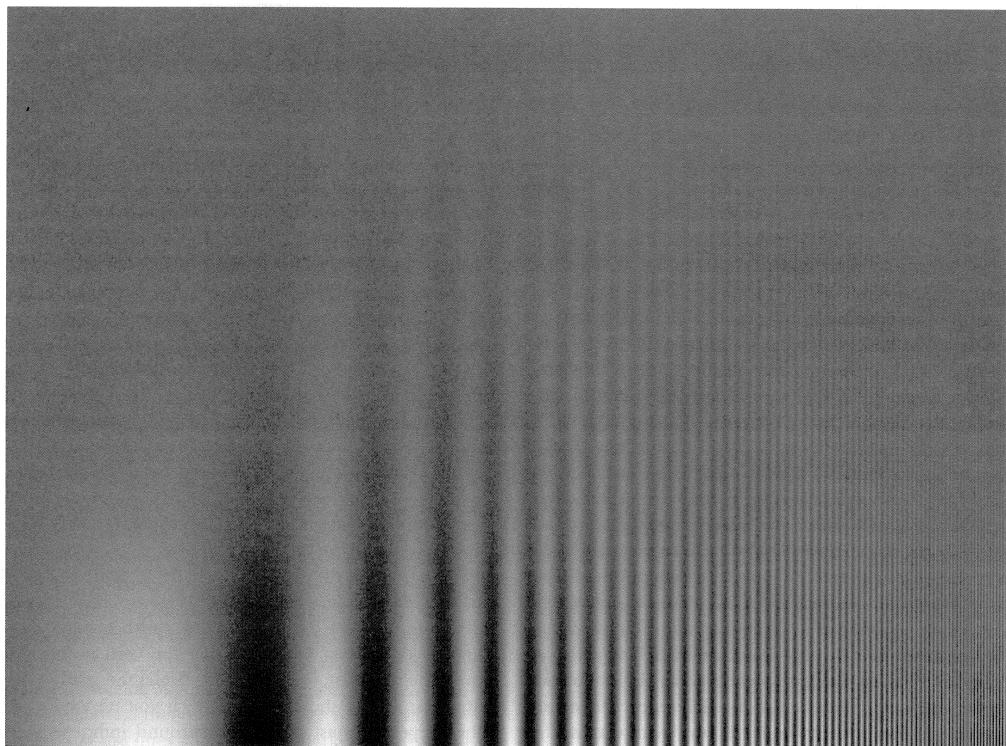


Figura 71: Sensibilità al contrasto: in questa figura la frequenza spaziale di una sinusoide aumenta da sinistra a destra mentre il contrasto diminuisce verso l'alto. Vediamo facilmente che la nostra sensibilità al contrasto dipende dalla frequenza spaziale. L'area nella quale si distingue la sinusoide è schematizzata in figura 72.



Figura 73: Un'immagine e le sue riproduzioni ottenute usando solo le frequenze di Fourier "basse" o solo quelle "alte".

Fourier come a delle coordinate di una particolare immagine nello spazio delle immagini dato che questi coefficienti la individuano univocamente.

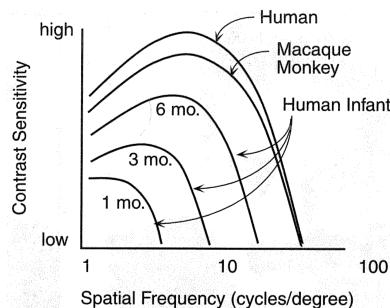


Figura 72: Sensibilità al contrasto al variare dell'età e per il macaco.

Figura 72: Sensibilità al contrasto al variare dell'età e per il macaco.

L'approccio della frequenza spaziale delle immagini, proposto dagli psicofisici negli anni del 1960, sostiene che nella visione viene usata precisamente questa rappresentazione per le immagini. Adottando questo punto di vista si trova una interpretazione soddisfacente per il ruolo di molti tipi di neuroni presenti nel sistema visivo.

Vediamo per sommi capi co-

me si può usare un neurone con uscita lineare

$$y = \vec{x} \cdot \vec{w}$$

per calcolare i coefficienti di Fourier per una funzione $f(x)$ nel più semplice caso unidimensionale. Sappiamo che, nel caso discreto, l' n -esimo coefficiente di Fourier si calcola a partire da

$$C_n \propto \int f(x) \sin(nx) dx$$

che può essere letto come il prodotto scalare (in uno spazio a infinite dimensioni) fra le due funzioni $f(x)$ e $\sin(nx)$.

Ne discende che un neurone, che abbia i pesi w che dipendano da un ipotetica coordinata x con $\sin(nx)$, quando vede sugli ingressi i valori di $f(x)$, producendo il prodotto scalare fra $f(x)$ e i pesi, cioè $\sin(nx)$, di fatto calcola in uscita l' n -esimo coefficiente di Fourier. Un certo numero di questi neuroni che lavori in parallelo sugli input $f(x)$ produce i coefficienti di Fourier della

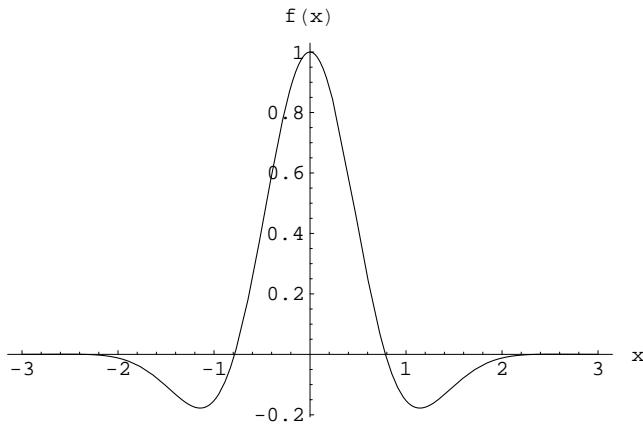


Figura 74: Grafico di $e^{-x^2} \cos(2x)$.

funzione $f(x)$. Analogamente, passando alle immagini, avendo diversi neuroni ognuno con le sinapsi che rappresentano una sua propria frequenza e direzione troveremo, al presentarsi di un’immagine sulla retina, le sue varie componenti di Fourier.

In realtà non si osservano neuroni con le sinapsi i cui valori seguano un andamento sinusoidale ma abbiamo visto che tipicamente la risposta spaziale delle sinapsi è a forma di cappello messicano. Però le funzioni trigonometriche non sono le uniche funzioni base per lo sviluppo in serie di Fourier: esistono altre funzioni che si rivelano più adatte al caso di funzioni non periodiche come sono tipicamente quelle che rappresentano immagini. Per esempio esistono le funzioni di Gabor che sono delle funzioni trigonometriche moltiplicate per una Gaussiana (e le loro parenti strette, le cosiddette “wavelet” functions). In questo caso la funzione base è sempre un seno o un coseno ma con l’ampiezza modulata da una Gaussiana e dunque con output non nullo solo in un ristretto intervallo; per esempio in figura 74 c’è il plot della funzione $e^{-x^2} \cos(2x)$. Si può dimostrare facilmente che si possono analizzare immagini in termini di serie di funzioni di Gabor o wavelet e, guarda guarda, il profilo tipico di queste funzioni ricorda molto da vicino il campo recettivo di un neurone visivo, figura 65.

Dunque questo approccio fornisce una interpretazione per i campi recettivi a cappello messicano ed anche a quelli di neuroni complessi della corteccia che hanno campi recettivi con più lobi, come in figura 66, e che corrisponderebbero a funzioni di Gabor con frequenza del coseno più elevata.

Un interessante esperimento, che sembra confermare questa teoria, è quello proposto da Campbell e Robson nel 1968, nel quale si misura il contrasto minimo necessario per vedere una immagine con una frequenza spaziale sinusoidale, come quella di figura 71. Se si ripete l’osservazione per un’onda quadra, una serie di righe bianche e nere, di uguale frequenza spa-

ziale sarebbe intuitivo aspettarsi che ci sia maggior sensibilità al contrasto per un'onda quadra che ha i bordi più netti ma invece, sorprendentemente, si trova che la sensibilità al contrasto è la stessa per l'onda sinusoidale e per l'onda quadra. Ma questo è esattamente il risultato previsto dalla nostra teoria. Infatti una funzione sinusoidale $\sin x$ verrebbe trasformata dal sistema visivo in un unico coefficiente di Fourier non nullo $(1, 0, 0, 0, \dots)$ corrispondente alla cosiddetta *frequenza fondamentale*. Un'onda quadra invece, come si vede dalla (49) ha coefficienti $\frac{4}{\pi}(1, 0, \frac{1}{3}, 0, \frac{1}{5}, 0, \dots)$ e si differenzia dal seno a partire dal terzo coefficiente che però vale $\frac{1}{3}$. Se immaginiamo che l'ampiezza del segnale debba superare una certa soglia per essere rivelato è ragionevole aspettarsi che questa riguarderà l'ampiezza maggiore che però risulta praticamente identica per il seno o l'onda quadra e corrispondente alla frequenza fondamentale. Ne consegue che il seno e l'onda quadra hanno la stessa soglia per essere rilevate e questo è esattamente quanto viene osservato sperimentalmente.



Figura 75: David Marr (1945–1980)

13.3 Approccio computazionale: il riconoscimento dei bordi

David Marr (figura 75), pur scomparso prematuramente, ha contribuito moltissimo allo studio della visione, ed ha proposto un modello per i sistemi informativi (biologici e non) basato su 3 livelli, sviluppato successivamente dal gruppo del MIT:

- computazionale — descrive relazioni fra input e output del sistema informativo senza specificare come ottenerle;
- algoritmico — descrive gli algoritmi che realizzano le operazioni descritte al livello superiore;
- implementazione — descrive il sistema fisico che realizza gli algoritmi.

Nell'analizzare il riconoscimento dei bordi rimarremo ai due livelli inferiori e studieremo solo una minuscola tessera nel grande mosaico della visione. Marr sostiene inoltre che la visione, data l'enorme quantità di informazione che deve manipolare, ha la necessità di abbandonare l'immagine fisica il prima possibile sostituendola con una sua rappresentazione simbolica (per esempio: una quercia in mezzo a un prato invece dei pixels che la ritraggono) la quale può essere manipolata molto più facilmente. Secondo Marr

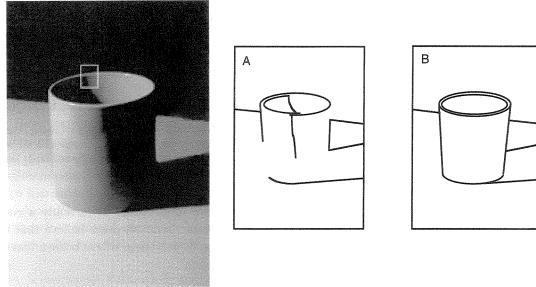


Figura 76: Un’immagine in bianco e nero con i bordi “veri” e come li vediamo.

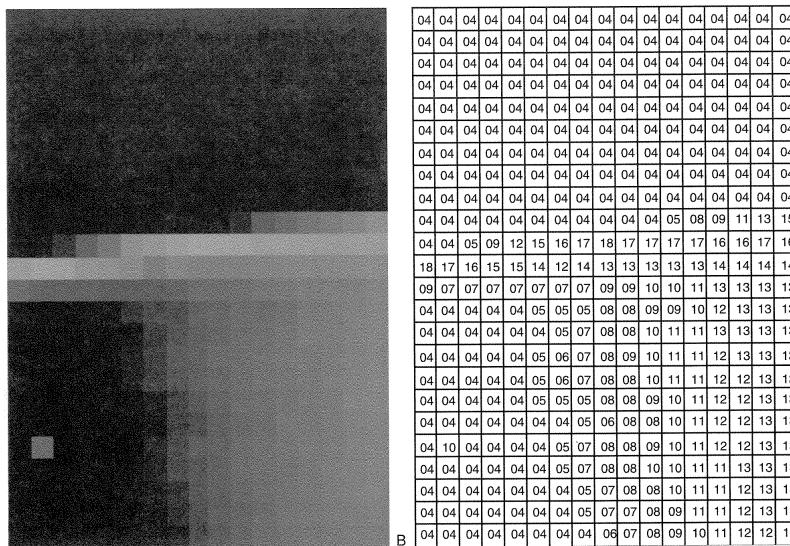


Figura 77: Dettaglio nel quadratino della figura 76 e in forma numerica.

l'immagine simbolica è formata da: bordi, linee, "blobs" e "terminazioni" legati fra loro da varie relazioni. Chiaramente questo approccio andrà bene per immagini naturali ma assai peggio per immagini che contengano solo rumore (per esempio quello che si vede alla TV non sintonizzata su alcun canale).

Per seguire la traccia di Marr e trasformare l'immagine in simboli, visto il tipo di immagini con le quali abbiamo a che fare (e che, ricordiamolo, sono un sottinsieme estremamente particolare di tutte le immagini possibili) una delle prime cose da fare è riconoscere nelle immagini i bordi che segnalano diversi oggetti che fanno parte dell'immagine o diverse illuminazioni (per esempio le ombre). Riconosciuti i bordi potremo fare una primissima descrizione simbolica dell'immagine che conterrà una lista di bordi e di aree uniformi.

Per semplificare le cose supponiamo di avere a che fare con immagini in bianco e nero come quella nelle figure 76 e 77, che matematicamente idealizziamo come una matrice di numeri, nella quale dobbiamo riconoscere dei bordi. Osserviamo dalle due immagini piccole di figura 76 che una cosa sono i bordi veri dell'immagine ed una cosa sono i bordi che il nostro sistema visivo ci propone; noi affrontiamo il problema più elementare di trovare i bordi veri dell'immagine. La prima idea per realizzare un riconoscitore di bordi può essere quella di fare la differenza fra due pixel adiacenti e ripetendo l'operazione per ogni pixel generare una nuova immagine che in ogni pixel contiene questa differenza. La differenza di due pixel equivale a considerare un operatore $-+$ applicato ad ogni pixel dell'immagine (tralasciamo i problemi dei bordi). Quest'operatore di fatto calcola la derivata spaziale dell'immagine e il suo risultato sarà 0 in zone "piatte" dell'immagine originale e diverso da 0 nei punti di variazione, cioè nei bordi.

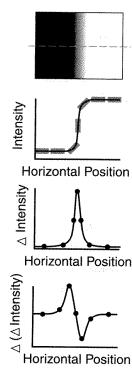


Figura 78: Bordo e andamento dell'intensità e delle sue prime due derivate.

Dovremmo calcolare però una differenza di questo tipo per ognuna delle 8 direzioni nelle quali ci possiamo muovere partendo da un dato pixel e dunque produrre 8 nuove immagini ognuna delle quali conterebbe solo i bordi ortogonali alla particolare direzione della differenza come si vede in figura 78.

Invece di usare un operatore $-+$ per ogni direzione si può pensare di creare un nuovo operatore che sia la somma degli 8 operatori. Esso si presenterà come

$$\begin{matrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{matrix} \quad (50)$$

ed è un operatore senza una particolare preferenza direzionale che produce una sola immagine a partire da quella grezza. Questo operatore in realtà esegue la derivata seconda spaziale (maggiori dettagli nel paragrafo 13.3.1) e in esso i bordi sono segnalati da un doppio picco e coincidono con il punto nel quale la derivata seconda vale 0 come schematizzato in figura 78. È facile immaginare di implementare una manipolazione di questo genere dell'immagine grezza con una rete neurale: per prima cosa pensiamo ai recettori come ad una matrice di neuroni ai quali è collegato un secondo strato con uguale numero di neuroni. Ogni neurone del secondo strato è collegato con peso 8 al suo input corrispondente e con peso -1 agli 8 neuroni di input adiacenti. In questo modo, in un solo passaggio, si passa dall'immagine grezza del primo strato a quella ottenuta applicando l'operatore (o più propriamente: facendone la convoluzione): essa è l'immagine che si forma al secondo strato della rete.

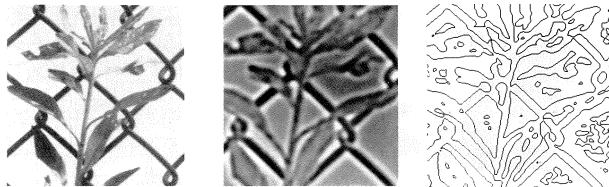


Figura 79: A sinistra un'immagine in bianco e nero, in mezzo l'immagine prodotta dalla convoluzione con l'operatore (50), a destra i bordi rilevati.

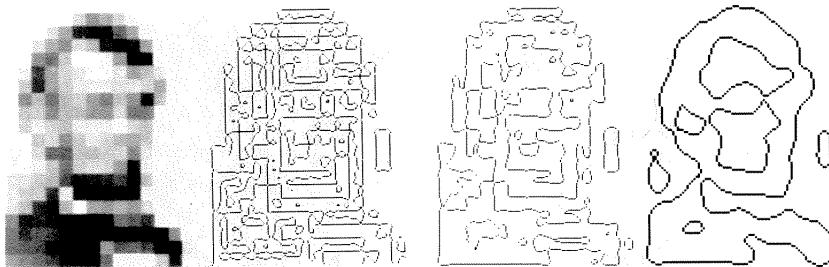


Figura 80: A sinistra immagine a risoluzione spaziale ridotta, nelle tre immagini a destra output prodotto da tre rivelatori di bordi a diversa risoluzione spaziale.

L'analisi delle connessioni della retina con la curva di risposta a cappello messicano ha indotto Marr a ipotizzare che esse sono parte, insieme ai livelli successivi, di un raffinato sistema di rilevazione di bordi e linee rette e non può non colpire la somiglianza della risposta a cappello messicano con l'operatore (50) trovato con semplici considerazioni, vedi figura 81. Nella figura 79 si vede un'immagine e l'immagine ottenuta dall'applicazione dell'operatore (50); in questo caso i bordi sono segnalati da dei punti di 0 dell'immagine che corrispondono ai grigi neutri; nella terza immagine sono segnati solo i bordi.

Il passo successivo è definire operatori a diversa dimensione per catturare le diverse frequenze spaziali dell'immagine, per esempio la figura 80 contiene un'immagine a risoluzione ridotta e la risposta di tre rivelatori di bordi a risoluzione spaziale via via calante; solo nell'ultima immagine si riconoscono i tratti dell'immagine.

Con questo semplice esempio abbiamo voluto mostrare come sia possibile, con una semplice rete neurale, costruire un rivelatore di bordi, e come in questa struttura si possa vedere una certa somiglianza con quella della retina.

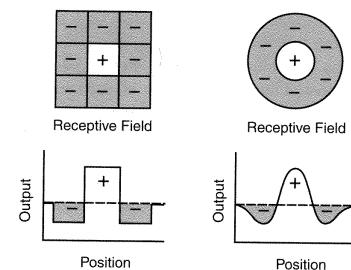


Figura 81: Versione discreta e continua dell'operatore (50) e profili di risposta lungo il diametro.

13.3.1 Calcolo numerico delle derivate

Qui diamo qualche dettaglio sull'operatore (50). Un problema frequente è quello di dover calcolare la derivata di una funzione $f(x)$ della quale conosciamo solo il valore numerico su alcuni punti $f(x_1), f(x_2), \dots, f(x_n)$, come per esempio nel caso delle immagini a pixel che possiamo pensare come il valore dell'intensità luminosa rilevato su un reticolo di punti. Se vogliamo stimare la derivata $f'_s(x)$ e abbiamo a disposizione solo questi valori numerici, la prima cosa che viene in mente di fare è:

$$f'_s(x_i) = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} = \frac{f(x_i + h) - f(x_i)}{h}$$

dove, per semplicità, abbiamo supposto che la distanza fra tutti i punti sia fissa ed uguale ad h . Per valutare l'errore che si commette espandiamo in serie di Taylor $f(x_i + h)$

$$f'_s(x_i) = \frac{f(x_i) + hf'(x_i) + \frac{h^2}{2}f''(x_i) + \dots - f(x_i)}{h} = f'(x_i) + \frac{h}{2}f''(x_i) + \dots$$

Una stima migliore di $f'(x_i)$ si ottiene usando la media delle due differenze calcolate rispetto a $f(x_{i-1})$ e $f(x_{i+1})$

$$f'_s(x_i) = \frac{1}{2} \left(\frac{f(x_i) - f(x_{i-1})}{h} + \frac{f(x_{i+1}) - f(x_i)}{h} \right) = \frac{f(x_{i+1}) - f(x_{i-1})}{2h}$$

e usando lo sviluppo in serie di Taylor facilmente si trova che in questo caso

$$f'_s(x_i) = f'(x_i) + \frac{h^2}{6}f'''(x_i) + \dots$$

dunque un approssimazione di un ordine superiore della derivata vera in quanto l'errore sarà in questo caso dell'ordine di h^2 .

Per il nostro problema di immagini questo stimatore vorrebbe dire fare la convoluzione con l'operatore asimmetrico $-\frac{1}{2} \quad 0 \quad \frac{1}{2}$ e se lo sommassimo per ottenere la derivata in tutte le direzioni troveremmo un operatore totalmente nullo.

Per aggirare questa difficoltà vediamo come si può stimare numericamente la derivata seconda: consideriamo lo stimatore

$$f''_s(x_i) = \frac{2f(x_i) - f(x_{i+1}) - f(x_{i-1})}{h^2}$$

e, mediante sviluppi in serie di Taylor, troviamo che fornisce un'approssimazione

$$f''_s(x_i) = -f''(x_i) - \frac{h^2}{12}f''''(x_i) + \dots$$

e dunque la convoluzione dell'immagine con l'operatore $-1 \quad 2 \quad -1$ produce uno stimatore (dell'opposto) della derivata seconda approssimato all'ordine di h^2 . Inoltre è un operatore simmetrico che può essere applicato contemporaneamente a tutte le direzioni e fornisce un'interpretazione più corretta dell'operatore (50).

13.4 Problemi mal posti

Il problema tipico della visione è ricostruire uno spazio tridimensionale a partire da una sua proiezione bidimensionale sulla retina e dall'esempio visto in figura 60 abbiamo capito intuitivamente la definizione di problema mal posto. Già a questo livello possiamo capire che questo problema non può essere risolto senza informazione aggiuntiva: per esempio che l'oggetto è una barra verticale di lunghezza l ; con questi dati il problema diventa risolubile. Nel caso dell'occhio non avremo informazioni aggiuntive di questo tipo ma potremo ricavarle con certe assunzioni su quello che *soltamente* si vede: in altre parole useremo informazioni sulla distribuzione delle immagini per supplire l'informazione mancante alla retina.

Un altro esempio di problema mal posto è il Perceptron: data una serie di esempi di solito esistono infiniti valori dei pesi per i quali gli esempi sono classificati correttamente: si tratta dunque di un problema mal posto. Una possibilità è di considerare il Perceptron che massimizza il margine (vedi capitolo ??) e in questo caso, con l'informazione aggiuntiva che la soluzione deve massimizzare il margine, la soluzione diventa unica.

In questo paragrafo illustreremo questa tecnica matematica, detta *regolarizzazione*, trattando un semplice esempio legato al problema della rilevazione dei bordi. Iniziamo con la definizione di problemi mal posti.

13.4.1 Definizione dei problemi ben posti

Tradizionalmente nel meccanismo della visione si riconoscono due tipi di problemi: il primo è il cosiddetto problema diretto ed è quello di calcolare, a partire da: uno spazio tridimensionale, una data illuminazione, le leggi dell'ottica e le caratteristiche dell'occhio, l'immagine che si proietta sulla retina. Il secondo problema, detto inverso, si occupa di ricostruire lo spazio tridimensionale a partire dall'immagine sulla retina. In termini più matematici se x descrive una scena del mondo tridimensionale e y la sua proiezione sulla retina il problema diretto è quello di trovare la funzione $f()$ tale che $y = f(x)$ e quello inverso di trovare la sua inversa $x = f^{-1}(y)$.

Un altro esempio di problemi diretti e inversi è quello della creazione di un suono corrispondente a una certa parola e la ricostruzione della parola a partire dal suono. Come è intuitivo i problemi diretti sono molto più semplici dei problemi inversi e di solito sono ben posti.

Formalmente diremo che il problema (diretto o inverso) di trovare $f()$ tale che $y = f(x)$ è *ben posto* se verifica tre condizioni:

- esistenza — per ogni x esiste $y = f(x)$ (con x e y che appartengono ai relativi domini);
- unicità — per ogni x, z $f(x) = f(z)$ se, e solo se, $x = z$;

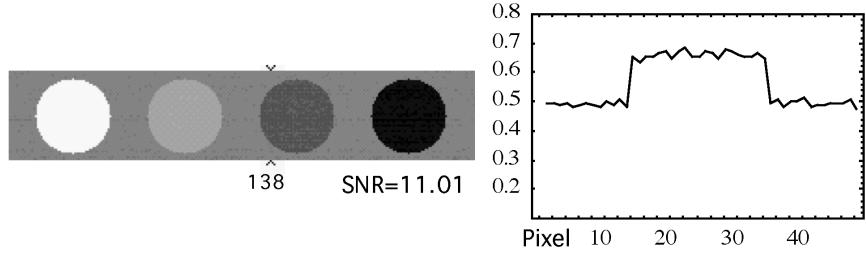


Figura 82: Immagine in bianco e nero con a destra il profilo lungo il bit 138.

- continuità (o stabilità) — dato x_0 per ogni ε esiste un δ tale che per ogni $|x - x_0| < \varepsilon$ vale $|f(x) - f(x_0)| < \delta$.

Se una sola delle tre condizioni risulta violata si dice che il problema è mal posto. Chiaramente il problema inverso della visione è mal posto perché viola, se non altro, la condizione di unicità.

13.4.2 Un esempio unidimensionale

Vediamo come *regolarizzare* un problema mal posto con un esempio molto semplice: supponiamo di avere una retina unidimensionale (la generalizzazione a una retina bidimensionale è immediata) che veda una superficie illuminata uniformemente ma che, a causa di rumore o altri fattori fisici, raccolga dei segnali diversi come nell'esempio fatto in figura 82: se ad un immagine del genere applicassimo un rivelatore di bordi vedrebbe un bordo quasi ad ogni pixel.

In questo caso il problema è di ricostruire la luminosità stimata y_i presente nell' i -esimo pixel a partire da quella rivelata x_i , cioè trovare una funzione $y_i = f(x_i)$. Chiaramente l'opzione $y_i = x_i \forall i$ è una possibilità ma non tiene conto del fatto che sappiamo che di solito le immagini naturali contengono grandi aree uniformi cioè che i bordi sono rari. Potremmo allora decidere di aggiungere un vincolo al nostro problema con la richiesta che i valori ricostruiti di luminosità y_i siano simili fra pixel adiacenti. Chiediamo allora che gli y_i siano quelli che minimizzano la quantità

$$H = \frac{1}{2} \left(\sum_{j=1}^n (y_j - x_j)^2 + \lambda \sum_{j=2}^n (y_j - y_{j-1})^2 \right)$$

che ha due contributi: il primo è minore, minore è la differenza fra segnale rivelato x e segnale ricostruito, il secondo è minore, minore è la differenza fra valori adiacenti ricostruiti. La costante λ regola il peso relativo dei due fattori. Chiaramente trattandosi di una forma quadratica il suo minimo si potrà trovare dall'azzerarsi delle derivate per cui

$$\frac{\partial H}{\partial y_i} = (y_i - x_i) + \lambda((y_i - y_{i-1}) - (y_{i+1} - y_i)) = y_i + 2\lambda y_i - x_i - \lambda(y_{i-1} + y_{i+1})$$

e richiedendo l'annullarsi delle derivate si trova

$$y_i = \frac{1}{1+2\lambda} (x_i + \lambda(y_{i-1} + y_{i+1}))$$

che per $\lambda = 0$ da la soluzione iniziale $y_i = x_i$ mentre per $\lambda \rightarrow \infty$ da $y_i \rightarrow \frac{y_{i-1} + y_{i+1}}{2}$ che è una soluzione completamente piatta.

Questa soluzione non tiene conto che ci possano essere dei *veri* bordi nell'immagine. Se vogliamo tener conto anche di questo fattore potremmo aggiungere una variabile b_i che valendo rispettivamente 1 o 0 segnala se sul pixel i -esimo c'è o no un bordo. Allora potremmo decidere di minimizzare la quantità

$$H = \frac{1}{2} \left(\sum_{j=1}^n (y_j - x_j)^2 + \lambda \sum_{j=2}^n (1 - b_j)(y_j - y_{j-1})^2 \right) + \mu \sum_{j=1}^n b_j$$

dove, rispetto a prima, minimizziamo la differenza fra pixel adiacenti solo se fra di essi non c'è un bordo e dove abbiamo aggiunto un ulteriore somma che conta il numero di bordi presenti nell'immagine ricostruita; il fattore μ regola il peso relativo del numero di bordi rispetto agli altri termini.

Senza addentrarci nello studio delle soluzioni di questo caso osserviamo il procedimento: abbiamo aggiunto alla somma base $\sum_{j=1}^n (y_j - x_j)^2$ dei termini che pesano la nostra conoscenza sul tipo di immagini che vogliamo ricostruire. In pratica se conoscessimo la distribuzione di tutte le immagini che questa retina deve ricostruire potremmo calcolare dei valori teorici per i coefficienti λ e μ che ci darebbero la massima fedeltà di ricostruzione della nostra retina.

Nel 1963 Tikhonov ha formalizzato questo approccio proponendo di cercare la soluzione $f()$ come quella che minimizza il funzionale

$$H(f) = \frac{1}{2} \left(\sum_{j=1}^n (f(x_j) - x_j)^2 + \lambda \|Df(x)\|^2 \right)$$

dove il primo termine tiene conto della riproduzione dei dati ed il secondo è un termine di *regolarizzazione* dove D indica un qualche operatore di differenziazione lineare che limita le oscillazioni eccessive di $f(x)$ e che tiene conto di informazioni che si possano avere sulla soluzione del problema. L'aggiunta di questo termine, controllato dal parametro di regolarizzazione λ ha lo scopo di rendere il problema ben posto, e quindi risolubile, restringendo l'insieme delle soluzioni possibili e quindi aggiungendo di fatto una conoscenza esterna sul tipo di soluzione cercata.

13.5 Altre interpretazioni della visione

Oltre a quanto visto nei precedenti paragrafi esistono anche altre possibili interpretazioni per le scoperte di Hubel e Wiesel sulla corteccia striata presentate nel paragrafo 13.1.3.

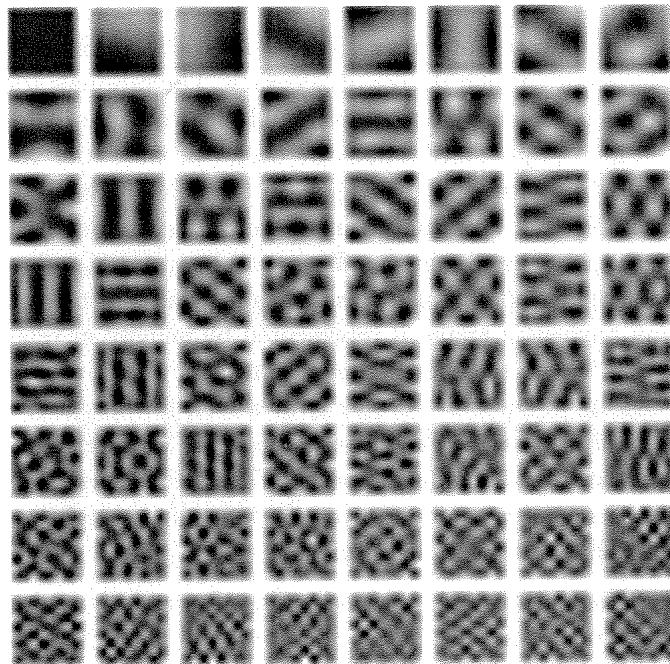


Figura 83: Campi recettivi delle componenti principali; i colori chiari indicano la presenza di sinapsi eccitatorie, quelli scuri sinapsi inibitorie, il grigio mancanza di connessioni.

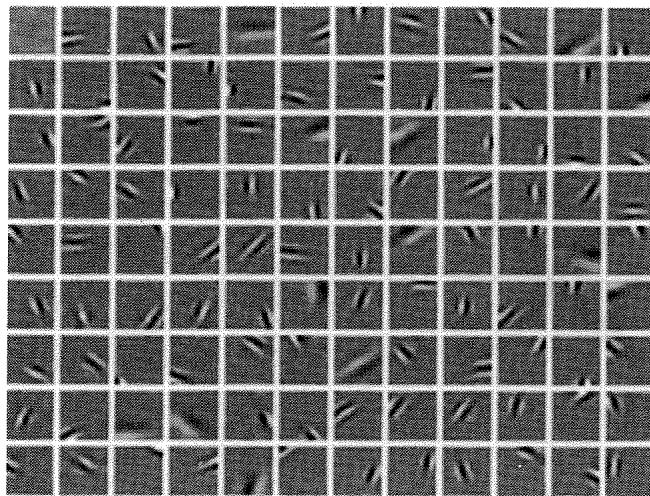


Figura 84: Campi recettivi ottenuti dalla richiesta di sparse distributed coding.

13.5.1 Interpretazioni basate sulle reti neurali

Iniziamo menzionando un approccio puramente fenomenologico nel quale si studia il sistema visivo cercando di far riprodurre ad una rete neurale artificiale alcune delle sue caratteristiche più salienti, per esempio ricostruire una superficie tridimensionale dalle ombre presenti in un'immagine (shape from shading). Per studiare questo problema si costruisce una rete di neuroni feed-forward nel quale si mette al primo livello una serie di neuroni con sinapsi fisse che seguono l'andamento a cappello messicano, uno strato intermedio ed uno strato di uscita cui si vuole far apprendere la forma tridimensionale dell'oggetto, opportunamente rappresentata. Generato un buon numero di esempi con le risposte corrette si esegue l'apprendimento con la back-propagation. Una volta ultimato l'apprendimento si esaminano le sinapsi dei neuroni che hanno appreso cercando di capire quale algoritmo realizzano e se ci sia qualche somiglianza con quelli biologici. Sorprendentemente si trova che le sinapsi trovate dall'apprendimento hanno caratteristiche qualitativamente estremamente simili a quelle delle reti biologiche.

13.5.2 Interpretazioni basate sulla distribuzione delle immagini naturali

Un altro approccio che è stato seguito per cercare di dare un'interpretazione al funzionamento della corteccia visiva parte dalla seguente ipotesi: visto che le immagini naturali sono un sottinsieme estremamente ridotto di tutte le immagini possibili e con proprietà statistiche molto particolari⁴⁰ ci si può aspettare che il sistema visivo sia adattato a queste immagini e ne sfrutti le proprietà per semplificare il proprio funzionamento.

Con questa ipotesi possiamo chiederci qual'è il sistema migliore, da un punto di vista matematico, per trattare le immagini: avendo noi imposto che le immagini seguano una certa distribuzione teorica abbiamo aggiunto dell'informazione al problema che così risulta *ben posto*, nel senso del paragrafo 13.4, e può essere affrontato matematicamente.

Resta da definire cosa si intende per sistema *migliore*, una prima definizione è questa: si cerca un sistema che possa codificare le immagini minimizzando il numero di informazioni trasmesse (compact coding). Questa richiesta appare sensata perché da la massima compressione per le immagini naturali e corrisponde alla richiesta di minima ridondanza incontrata per uno strato di Perceptron nel capitolo 10. Partendo da questa richiesta si trova, abbastanza facilmente, che la miglior soluzione è quella di codificare

⁴⁰ Le loro proprietà statistiche sono state studiate da Field dal 1993 che ha osservato che possono essere ragionevolmente rappresentate da una distribuzione tipo quella di Boltzmann (38) (sorpresa !) nella quale l'energia è rappresentata dalla dissimilarità fra pixel adiacenti. Questo fa sì che le immagini naturali più probabili siano quelle dotate di grandi aree uniformi. Detto in altre parole la trasformata di Fourier spaziale delle immagini naturali è piccata a frequenza nulla.

le immagini secondo le loro *componenti principali*, descritte nell'appendice A.4. Per esempio considerando una retina di 8×8 pixel, cui è collegato un secondo strato di 8×8 neuroni, ognuno connesso a tutti i 64 elementi della retina, si trova che, i pesi del secondo strato che fanno sì che un'immagine sulla retina possa essere rappresentata dal minor numero di neuroni al secondo strato, sono le componenti principali delle immagini. In figura 83 sono rappresentate le sinapsi dei neuroni del secondo strato in corrispondenza dei 64 recettori sulla retina ottenute dopo la presentazione sulla retina di un certo numero di immagini naturali e il conseguente apprendimento: i colori chiari indicano sinapsi eccitatorie mentre quelli scuri inibitorie. Viste le proprietà delle componenti principali si ha che prendendo i primi k neuroni del secondo strato avremo la miglior rappresentazione (nel senso di varianza spiegata) a k dimensioni delle nostre immagini. Il risultato è interessante però non ci sono molti neuroni nella corteccia visiva che abbiano campi recettivi simili a quelli rappresentati in figura 83.

Un risultato diverso si ottiene imponendo la richiesta che, per ogni immagine, nel secondo strato sia minimo il numero di neuroni attivi contemporaneamente (sparse distributed coding). Con questa richiesta, un set di immagini naturali simili al caso precedente, fa trovare ad un algoritmo di apprendimento non supervisionato, che agisce su una retina di 9×12 recettori, le sinapsi di figura 84 che, adesso sì, assomigliano ai campi recettivi osservabili nella corteccia primaria (vedi figura 69) infatti vi troviamo neuroni che reagiscono a direzioni privilegiate situate in particolari posizioni della retina.

Più precisamente questo risultato è stato ricavato dalle richieste contemporanee che la mutua informazione fra la retina ed il primo strato fosse massima, cioè che nel secondo strato vi fosse tutta l'informazione presente nel primo, e che, per ogni immagine, il numero di neuroni attivi nel secondo strato fosse minimo. Mentre la prima richiesta è ovvia e l'abbiamo già incontrata nel capitolo 10, la seconda lo è di meno ma queste sono alcune delle sue possibili giustificazioni:

- se pochi neuroni sono attivi contemporaneamente migliora il rapporto segnale rumore;
- è più facile memorizzare e ritrovare le immagini in una memoria associativa se pochi neuroni sono attivi per ogni pattern;
- c'è evidenza fisiologica che nei sistemi biologici solo qualche % dei neuroni è attivo contemporaneamente;
- una rete successiva che analizzi le caratteristiche delle immagini avrà il compito facilitato se pochi neuroni sono attivi contemporaneamente.

13.5.3 Ipotesi dei canali separati

Infine menzioniamo l’ipotesi dei canali separati nell’interpretazione del sistema visivo (visual pathways). Secondo questa ipotesi esistono cammini paralleli e separati per le analisi delle immagini: uno ciascuno per forma, moto, colore e visione binoculare (profondità) che sono psicologicamente integrati ma percepiti separatamente. A supporto di questa ipotesi vi sono le osservazioni che l’integrazione può essere controllata dall’attenzione e che esistono soggetti che possono perdere selettivamente una delle capacità visive, per esempio la rilevazione del moto.

13.6 Il “Cappello Messicano”

Spendiamo due parole sul campo recettivo a cappello messicano che abbiamo incontrato molte volte in questo capitolo.

- Per prima cosa ricordiamo l’evidenza biologica dove, seppure in forme leggermente diverse, il campo recettivo a cappello messicano compare in tutti gli stadi della visione: dalla retina alla corteccia striata (paragrafi 13.1.1–13.1.3);
- è apparso come una delle funzioni base per l’espansione delle immagini in funzioni di Gabor (paragrafo 13.2);
- è apparso, in forma discretizzata, come operatore per fare la convoluzione delle immagini nella teoria computazionale (paragrafo 13.3);
- è apparso, seppure in forma leggermente modificata, nei campi recettivi che si trovano applicando la richiesta di sparse distributed coding (paragrafo 13.5.2);
- infine abbiamo visto che appare quando si sviluppa l’algoritmo di massimizzazione della mutua informazione per uno strato di Perceptron binari con $\alpha > 1$ (capitolo 10).

Tutti questi casi hanno proposto possibili interpretazioni all’evidenza fisiologica che i campi recettivi nella visione hanno questo aspetto caratteristico. Capire quale di queste diverse interpretazioni (o anche altre) sia la più verosimile è un obiettivo per la ricerca futura.

14 Conclusioni

Abbiamo presentato diversi studi/algoritmi visti come metodi di programmazione di macchine massicciamente parallele, cioè reti di neuroni di McCulloch e Pitts con l'idea che ci sia un unico programma per tutte le unità (neuroni). I problemi che usualmente si trattano con le reti neurali sono soprattutto problemi che risultano difficili da programmare con macchine di tipo tradizionale come per esempio: pattern recognition, memorie associative etc. Questo perché, apparentemente, le reti potrebbero portare più facilmente alla soluzione.

Il nostro punto di vista è stato però complementare cioè abbiamo cercato di mostrare come le reti di neuroni possano trattare problemi che sono ben risolubili tradizionalmente: fit di una retta, calcolo delle componenti principali dei dati, sorting etc. In questo modo abbiamo cercato di mostrare che iniziamo a capire come programmare le reti anche in casi conosciuti e nei quali possiamo confrontare quantitativamente vantaggi e svantaggi dei due approcci.

Infine una nota sulla matematica: in questo corso se n'è usata parecchia ma appare sempre più chiaro che per avanzare nel campo delle reti neurali occorre insistere in questa direzione. È finito il tempo (se mai c'è stato) in cui bastava una rete simulata con 100 righe di programma per fare scoperte interessanti ed è anche al tramonto il periodo delle ricette euristiche. Senza l'*insight* fornito da un analisi matematica approfondita non c'è speranza di avanzare, per cui, il mio consiglio è: studiate matematica, studiate matematica, studiate matematica, e poi applicatela al vostro problema.

In questa dispensa abbiamo privilegiato una trattazione matematica che ha prodotto deduzioni giustificabili; non abbiamo parlato di argomenti magari più affascinanti ma meno sostenibili come ad esempio: se ad una memoria associativa aggiungo ai pesi un termine $e^{-\lambda t}$ ottengo una memoria che dimentica nel tempo e che potrebbe fornire un modello per la memoria a breve termine. È stata una scelta precisa.

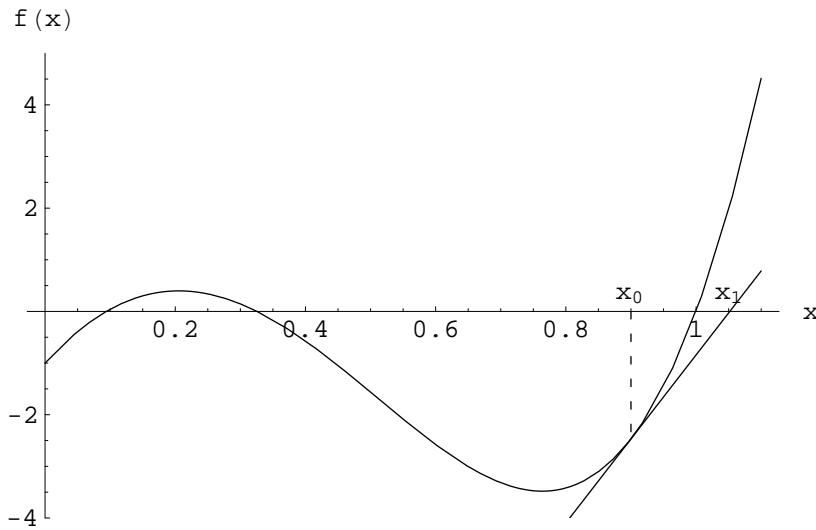


Figura 85: Metodo di Newton per trovare gli zeri di una funzione.

A Appendici

A.1 Metodi di minimizzazione

Consideriamo il problema di minimizzare una funzione come potrebbe essere la funzione d'errore più volte incontrata. Per brevità studiamo il caso più semplice della minimizzazione di $y = f(x)$ e cioè quello di una funzione che dipenda da una sola variabile e che sia derivabile e con derivata continua in tutto l'intervallo di interesse. In questo caso il problema di trovare gli estremi della funzione corrisponde al problema di trovare gli zeri della derivata, cioè le soluzioni di $f'(x) = 0$ per cui ci si riconduce al problema della ricerca degli zeri di una funzione e da questo problema iniziamo.

Il metodo iterativo proposto da Newton per calcolare numericamente gli zeri di una funzione $f(x)$, riprodotto in figura 85, deriva dall'equazione della retta tangente alla curva $y = f(x)$ calcolata nel punto x_0 che facilmente si trova essere

$$y = f'(x_0)(x - x_0) + f(x_0)$$

Risolvendo per $y = 0$ da questa si ricava il valore di x_1 corrispondente

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} \quad (51)$$

che può essere interpretata in questo modo: x_1 è la soluzione dell'equazione $f(x) = 0$ quando si approssimi la curva $y = f(x)$ con la sua derivata calcolata nel punto x_0 . La soluzione così trovata viene poi usata come nuovo punto di partenza e l'algoritmo viene iterato. In sostanza l'algoritmo di Newton si può schematizzare così:

1. si sceglie un punto iniziale x_0 come soluzione approssimata dell'equazione $f(x) = 0$;
2. si calcola un'approssimazione successiva della soluzione con la (51);
3. si termina se la precisione voluta è stata raggiunta altrimenti si ritorna al punto 2 sostituendo x_1 al posto di x_0 .

Si può dimostrare che, in casi non patologici, questo algoritmo converge molto rapidamente alla soluzione cercata (vedi su [https://en.wikipedia.org/wiki/Newton's method](https://en.wikipedia.org/wiki/Newton's_method) con una bella animazione del metodo).

Il metodo può essere banalmente esteso a cercare massimi e minimi di una funzione cercando zeri della sua derivata, ovviamente in questo caso la (51) diventa

$$x_1 = x_0 - \frac{f'(x_0)}{f''(x_0)}$$

e permette di trovare efficientemente le coordinate dei valori estremi di una funzione se conosciamo le sue prime due derivate.

Un metodo più semplice per trovare i minimi di una funzione e basato su principi simili, è il cosiddetto metodo della *discesa lungo il gradiente* nel quale si parte dallo sviluppo della funzione

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2}f''(x_0)(x - x_0)^2 + \dots$$

dalla quale

$$f(x) - f(x_0) = f'(x_0)(x - x_0) + O((x - x_0)^2)$$

e se scegliamo x in modo che

$$x - x_0 = -\varepsilon f'(x_0)$$

con ε piccolo e positivo e, trascurando i termini di ordine superiore al primo, si ha

$$f(x) - f(x_0) = -\varepsilon f'(x_0)^2 \leq 0$$

dalla quale si ricava

$$f(x) \leq f(x_0)$$

Il tutto si può così riassumere: se partiamo da x_0 e ci spostiamo di una quantità piccola $x = x_0 - \varepsilon f'(x_0)$ al nuovo punto il valore della funzione sarà minore o uguale al valore di partenza. In sostanza abbiamo calcolato la derivata della funzione in un punto e ci siamo spostati nella direzione di derivata negativa per cui, al limite di $\varepsilon \rightarrow 0$, siamo sicuri di arrivare in un punto dove la funzione ha un valore minore. La definizione di discesa lungo il gradiente deriva dall'idea di spostarsi a piccoli passi nella direzione

nella quale la funzione è decrescente (per cercare massimi basta, ovviamente, salire lungo il gradiente).

Questo algoritmo si estende immediatamente al caso di funzioni $f : \mathbb{R}^n \rightarrow \mathbb{R}$, in questo caso basta applicare il metodo ad ogni coordinata x_i , in pratica si ottiene il gradiente della funzione $\vec{\nabla}f$. In sostanza l'algoritmo di discesa lungo il gradiente si può schematizzare così:

1. si sceglie un punto iniziale \vec{x}_0 ;
2. si calcola un punto successivo con $\vec{x}_1 = \vec{x}_0 - \varepsilon \vec{\nabla}f(\vec{x}_0)$;
3. si termina se il gradiente era nullo altrimenti si sostituisce \vec{x}_1 al posto di \vec{x}_0 e si ritorna al punto 2.

Questo algoritmo non è così efficiente come quello di Newton ma è più semplice da applicare in quanto richiede solo la conoscenza della derivata prima della funzione.

Osserviamo infine che entrambi i metodi hanno bisogno di un valore iniziale di x_0 e da questo conducono al minimo più vicino. Per cui, se la nostra funzione ha diversi minimi relativi, questi metodi tendono a condurci al minimo relativo più vicino al punto di partenza il quale, in generale, non coinciderà con il minimo assoluto della funzione. Un parziale rimedio a questo problema può essere di far girare più volte l'algoritmo facendolo iniziare da punti diversi e dando come risposta finale il migliore dei minimi trovati.

A.2 La distribuzione binomiale

Se lanciando una moneta la probabilità di avere testa è p , e di conseguenza quella di avere croce è $q = 1 - p$, la probabilità $P(k, N)$ di ottenere k teste in N lanci (o nel lancio di N monete) è

$$P(k, N) = \binom{N}{k} p^k q^{N-k}$$

e il nome e le proprietà di questa distribuzione derivano dallo sviluppo del binomio

$$(p + q)^N = \sum_{k=0}^N \binom{N}{k} p^k q^{N-k}$$

e, dall'analisi del caso $p = q = 1/2$ si trova

$$(p + q)^N = 1 = \sum_{k=0}^N \binom{N}{k} \frac{1}{2^k} \frac{1}{2^{N-k}} = \frac{1}{2^N} \sum_{k=0}^N \binom{N}{k}$$

che prova la utile relazione

$$\sum_{k=0}^N \binom{N}{k} = 2^N$$

Il numero medio di teste si calcola con il valore di aspettazione di k

$$E[k] = \sum_{k=0}^N k P(k, N) = \sum_{k=0}^N k \binom{N}{k} p^k q^{N-k}$$

e, visto che il generico termine della somma si può scrivere come

$$k \binom{N}{k} p^k q^{N-k} = p \frac{\partial}{\partial p} \binom{N}{k} p^k q^{N-k}$$

ricaviamo

$$\begin{aligned} E[k] &= \sum_{k=0}^N p \frac{\partial}{\partial p} \binom{N}{k} p^k q^{N-k} = p \frac{\partial}{\partial p} \sum_{k=0}^N \binom{N}{k} p^k q^{N-k} = \\ &= p \frac{\partial}{\partial p} (p+q)^N = p N (p+q)^{N-1} = N p \end{aligned}$$

Con tecnica analoga si può calcolare la varianza di k che risulta

$$\text{Var}[k] = E[k^2] - E[k]^2 = \dots = Np(1-p)$$

Infine osserviamo che per N sufficientemente grandi la distribuzione binomiale è molto ben approssimata dalla distribuzione Gaussiana (cui essa tende per $N \rightarrow \infty$) che indichiamo con

$$G_{\mu\sigma}(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \quad (52)$$

e per approssimare la binomiale la Gaussiana dovrà avere parametri $\mu = Np$ e $\sigma = \sqrt{Np(1-p)}$, vedi figura 86. In pratica approssimeremo

$$P(k, N) \simeq \int_{k-\frac{1}{2}}^{k+\frac{1}{2}} G_{\mu\sigma}(x) dx \simeq G_{\mu\sigma}(k)$$

che possiamo anche usare per il calcolo di coefficienti binomiali quando N è grande dato che facilmente si trova da $P(k, N)$ per $p = 1/2$

$$\binom{N}{k} \simeq \frac{2^{N+1}}{\sqrt{2\pi N}} e^{-\frac{1}{2}(\frac{2k-N}{\sqrt{N}})^2}$$

e approssimeremo somme di probabilità binomiali con un integrale della relativa Gaussiana. Ricordando la definizione della primitiva della Gaussiana

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

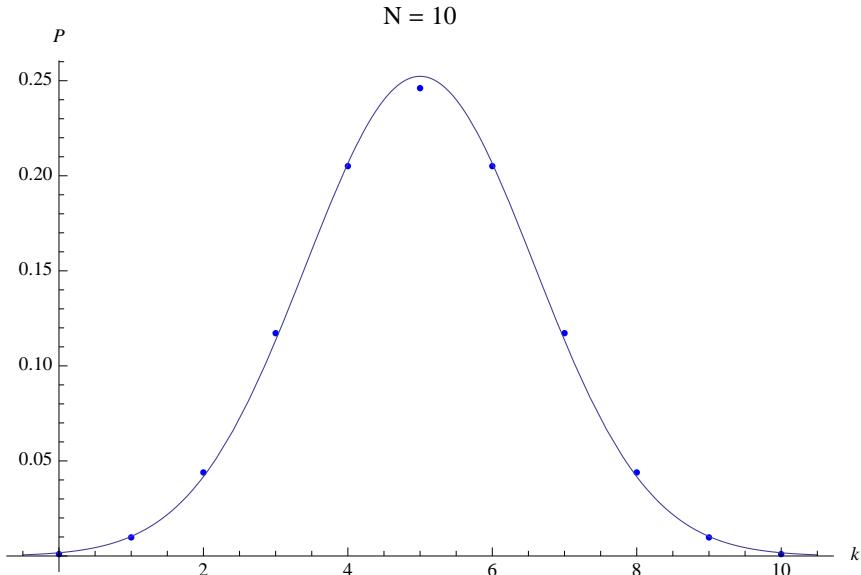


Figura 86: Approssimazione Gaussiana della binomiale con $N = 10$ e $p = 1/2$, si vede che già per N piccoli l'approssimazione è buona.

facilmente si ricava che

$$\int_{-\infty}^x G_{\mu\sigma}(t)dt = \frac{1}{2} \left[1 + \operatorname{erf} \left(\frac{x - \mu}{\sigma\sqrt{2}} \right) \right] \quad (53)$$

alla quale basterà sostituire i valori di x , μ e σ per ottenere l'approssimazione desiderata.

A.3 La matrice di covarianza

Quando si ha a che fare con dati a n componenti⁴¹ è utile definire la *matrice di covarianza* dei dati che generalizza il concetto di varianza. Per prima cosa il valor medio dei nostri dati è, in forma vettoriale,

$$\vec{\mu} = \frac{1}{m} \sum_{\nu=1}^m \vec{x}_\nu := \langle \vec{x} \rangle \quad \vec{x}_\nu \in \mathbb{R}^n$$

dove abbiamo usato la notazione introdotta al capitolo 9 per indicare la media sui dati. La varianza di una delle componenti è

$$\sigma_i^2 = \frac{1}{m} \sum_{\nu=1}^m (x_{i\nu} - \mu_i)^2 = \langle (x_i - \mu_i)^2 \rangle = \langle x_i^2 \rangle - \mu_i^2 \quad i = 1, \dots, n$$

⁴¹Per esempio un vettore \vec{x} per indicare persone potrebbe avere delle componenti come: peso, altezza, età, sesso, reddito, giro vita, numero di braccia, etc. etc.

mentre invece la *covarianza* fra le due componenti dei dati x_i e x_j si definisce⁴²

$$\text{cov}(x_i, x_j) := \frac{1}{m} \sum_{\nu=1}^m (x_{i\nu} - \mu_i)(x_{j\nu} - \mu_j) = \langle (x_i - \mu_i)(x_j - \mu_j) \rangle = \langle x_i x_j \rangle - \mu_i \mu_j .$$

con le quali si può definire la matrice quadrata, simmetrica ($\text{cov}(x_i, x_j) = \text{cov}(x_j, x_i)$), $n \times n$ delle covarianze che sulla diagonale ha le varianze delle componenti ($\text{cov}(x_i, x_i) = \sigma_i^2$).

Noi usualmente supporremo i dati a media nulla, $\langle \vec{x} \rangle = \vec{0}$, se non lo fossero, volendo, si può sempre ridefinirli togliendo la loro media infatti $\langle \vec{x} - \vec{\mu} \rangle = \vec{0}$; in questo caso la matrice delle covarianze coincide con

$$\frac{1}{m} \sum_{\nu=1}^m x_{i\nu} x_{j\nu} = \langle x_i x_j \rangle := C_{ij} \quad i, j = 1, \dots, n$$

che è il generico elemento della matrice C calcolata dai dati. Visto che $x_i x_j$ è anche il generico elemento della matrice $n \times n$ $\vec{x} \vec{x}^T$, in notazione matriciale possiamo scrivere

$$C = \langle \vec{x} \vec{x}^T \rangle = \frac{1}{m} X X^T \quad (54)$$

dove la seconda relazione si ottiene facilmente chiamando X la matrice di n righe ed m colonne formata dai nostri dati. Dalla sua definizione si vede che anche la matrice C è una matrice quadrata di ordine n ed è simmetrica dato che $C_{ij} = C_{ji}$. I valori sulla diagonale della matrice sono i valori medi del quadrato dei dati:

$$C_{ii} = \langle x_i^2 \rangle .$$

Dato che sia la matrice delle covarianze che C sono simmetriche ne segue che hanno tutti gli autovalori λ_i reali così come i relativi autovettori \vec{e}_i che formano un base ortonormale completa in \mathbb{R}^n . Indichiamo con E la matrice formata dagli autovettori che gode delle usuali proprietà degli autovettori ed autovalori

$$E^T E = E E^T = \mathbb{1} \quad C E = E \Lambda \quad E^T C E = \Lambda$$

dove Λ è una matrice diagonale i cui elementi sono gli n autovalori di C che, per comodità, supporremo ordinati in ordine decrescente.

⁴²la *correlazione* invece è la covarianza fra variabili standardizzate, cioè a media nulla e varianza unitaria, dunque

$$-1 \leq \left\langle \frac{x_i - \mu_i}{\sigma_i} \frac{x_j - \mu_j}{\sigma_j} \right\rangle \leq 1$$

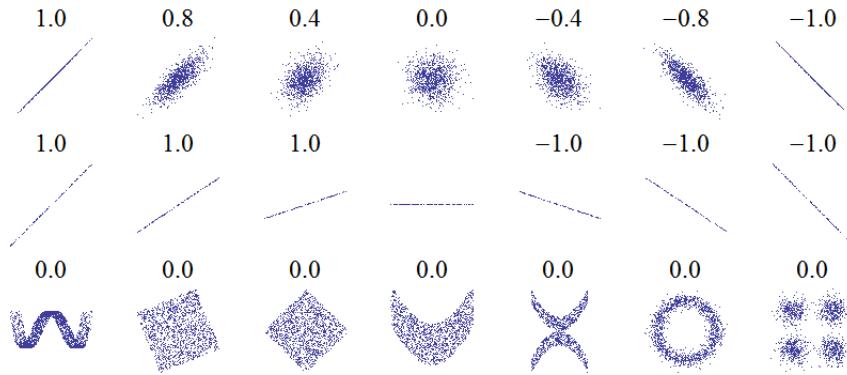


Figura 87: Dati distribuiti in due dimensioni con diversi valori di correlazione; si osservi che correlazione nulla non implica indipendenza.

Infine C è semidefinita positiva il che si dimostra facilmente provando che $\vec{z}^T C \vec{z} \geq 0 \quad \forall \vec{z}$, infatti

$$\vec{z}^T C \vec{z} = \vec{z}^T \langle \vec{x} \vec{x}^T \rangle \vec{z} = \langle \vec{z}^T \vec{x} \vec{x}^T \vec{z} \rangle = \langle (\vec{z}^T \vec{x})^2 \rangle \geq 0$$

e di conseguenza tutti i suoi autovalori sono non negativi: $\lambda_i \geq 0 \quad \forall i$.

Infine notiamo che la covarianza è un indicatore della dipendenza statistica fra diverse variabili ma mentre la indipendenza implica covarianza nulla, la covarianza nulla non implica indipendenza come appare in figura 87.

A.4 Il metodo delle componenti principali (PCA)

Quando si ha a che fare con dati \vec{x} con molte componenti viene spontaneo chiedersi se una qualche combinazione lineare delle varie componenti del vettore \vec{x} non sarebbe più significativa per studiare i dati. Matematicamente questo si può esprimere definendo una nuova variabile $y = \vec{w}^T \vec{x}$ dove il vettore \vec{w} contiene i coefficienti della combinazione lineare e la trasformazione è applicata allo spazio degli input.

Questi metodi si chiamano di proiezione e cercano una trasformazione degli input che massimizzi una qualche misura di “interesse” dei dati.

Per l’analisi delle componenti principali (Principal Component Analysis o [PCA](#)), proposta nel 1901 da Pearson e storicamente il primo metodo di questo tipo, la quantità di interesse che si cerca di massimizzare è la varianza dei dati. Dunque il metodo delle componenti principali cerca una *rotazione ortogonale* degli input che renda massima la varianza dei dati trasformati. Sia

$$y = \vec{w}^T \vec{x}$$

cercheremo \vec{w} in modo che si massimizzi la varianza di y . Supponiamo che la media degli \vec{x} sia nulla (il che si può comunque facilmente ottenere

sottraendo a ogni coordinata la sua media) e cioè

$$\frac{1}{m} \sum_{\nu=1}^m \vec{x}_\nu = \langle \vec{x} \rangle = \vec{0}$$

Per prima cosa calcoliamo il valor medio di y

$$\langle y \rangle = \left\langle \sum_{i=1}^n w_i x_i \right\rangle = \sum_{i=1}^n w_i \langle x_i \rangle = 0$$

dato che abbiamo supposto che i valori medi di x_i siano nulli. Di conseguenza la varianza di y sarà

$$\langle y^2 \rangle = \left\langle \sum_{i=1}^n \sum_{j=1}^n w_i x_i x_j w_j \right\rangle = \sum_{i=1}^n \sum_{j=1}^n w_i \langle x_i x_j \rangle w_j = \sum_{i=1}^n \sum_{j=1}^n w_i C_{ij} w_j = \vec{w}^T C \vec{w}$$

dove C è la solita matrice degli input che, per dati a media nulla, coincide con la matrice di covarianza.

Avendo calcolato la varianza di y , ora dobbiamo cercare il valore di \vec{w} che la massimizza. Prima di procedere osserviamo che se \vec{w} non è limitato possiamo rendere la varianza di y grande a piacere. Per cui dovremo massimizzare la varianza con la condizione aggiuntiva $\vec{w}^T \vec{w} = 1$. La ricerca di questo massimo si può fare con la tecnica dei moltiplicatori di Lagrange che ci danno la forma da massimizzare

$$\vec{w}^T C \vec{w} - \lambda(\vec{w}^T \vec{w} - 1)$$

derivando rispetto a \vec{w} e ponendo la derivata a zero troviamo

$$C \vec{w} - \lambda \vec{w} = (C - \lambda \mathbb{1}) \vec{w} = 0$$

dove $\mathbb{1}$ indica la matrice identità. Per risolvere questo sistema per prima cosa dobbiamo chiedere che il determinante $|C - \lambda \mathbb{1}| = 0$, altrimenti $\vec{w} = 0$ sarebbe l'unica soluzione possibile. Di conseguenza λ sono le soluzioni dell'equazione caratteristica della matrice, meglio note come autovalori della matrice, e i \vec{w} che annullano la derivata saranno quelli per i quali

$$C \vec{w} = \lambda \vec{w}$$

noti come autovettori della matrice. Sostituendo questo \vec{w} troviamo che $\langle y^2 \rangle = \vec{w}^T C \vec{w} = \lambda$ per cui la varianza di y sarà massimizzata quando λ è l'autovalore massimo della matrice e \vec{w} il corrispondente autovettore. Queste soluzioni esistono sicuramente dato che, come abbiamo visto nell'appendice A.3, la matrice C è simmetrica e semidefinita positiva ed è dotata di un insieme completo di autovettori ortonormali⁴³.

⁴³Notiamo che questo è equivalente a minimizzare la varianza nel sottospazio ortogonale a \vec{w} : dato che C è semidefinita positiva ha solo autovalori $\lambda_i \geq 0$ e massimizzare la varianza equivale a selezionare l'autovalore massimo e di conseguenza la somma degli autovalori rimanenti assume il valore minimo.

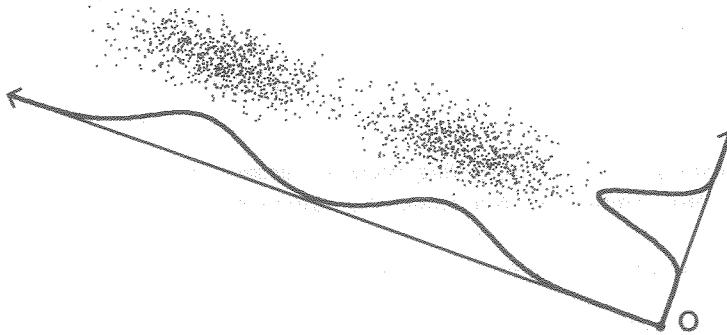


Figura 88: Dati distribuiti in due dimensioni con le due direzioni date dalle componenti principali e le distribuzioni dei dati lungo queste direzioni.

La ricetta del metodo delle componenti principali richiede che ora si trovi una direzione, ortogonale a quella appena trovata, che dia nuovamente la varianza massima. La condizione di ortogonalità è necessaria per avere una rotazione ortogonale e farà anche sì, come vedremo, che le varie coordinate “ruotate” y siano indipendenti fra loro.

Per giungere al punto finale, che le componenti principali sono gli n autovettori della matrice C , procediamo per induzione e supponiamo di aver trovato le prime $k - 1$ componenti principali e che esse corrispondano ai primi $k - 1$ autovettori della matrice e dimostriamo che la k -esima componente corrisponde al k -esimo autovettore. Dovremo cercare una direzione che massimizzi la varianza in una direzione ortogonale alle $k - 1$ direzioni già trovate per cui al solito massimeremo $\vec{w}^T C \vec{w}$ con le condizioni aggiuntive che il modulo di \vec{w} sia 1 e che siano rispettate le $k - 1$ condizioni di normalità. Questo problema si risolve, come il primo caso, con i moltiplicatori di Lagrange e il risultato è che la soluzione è l'autovettore di C corrispondente al k -esimo autovalore (supposti in ordine decrescente).

Riassumendo abbiamo visto che le condizioni di ortonormalità e di massimizzazione della varianza danno come direzioni di soluzione quelle determinate dagli autovettori della matrice C degli ingressi. Vediamo ora alcune proprietà di questa trasformazione introdotta sui nostri dati mentre in figura 88 è rappresentata una situazione ideale in due dimensioni:

- le n componenti principali sono

$$y_i = \vec{e}_i^T \vec{x} \quad i = 1, \dots, n$$

per cui, complessivamente, abbiamo introdotto una rotazione nello spazio degli input data da $\vec{y} = E^T \vec{x}$ dove E è la matrice degli autovettori di C ;

- le varianze delle coordinate trasformate $\langle y_i^2 \rangle = \lambda_i$ sono gli autovalori della matrice C ;
- le varie componenti principali danno la massima varianza nel sottospazio a loro disposizione (tolto cioè il sottospazio generato dalle componenti principali 'precedenti');
- ogni componente è scorrelata da tutte le altre infatti

$$\langle y_i y_j \rangle = \langle \vec{e}_i^T \vec{x} \vec{x}^T \vec{e}_j \rangle = \vec{e}_i^T \langle \vec{x} \vec{x}^T \rangle \vec{e}_j = \vec{e}_i^T C \vec{e}_j = \lambda_j \vec{e}_i^T \vec{e}_j = \delta_{ij} \lambda_j$$

in altre parole la matrice di covarianza per le componenti principali è la matrice diagonale degli autovalori, cioè $\langle \vec{y} \vec{y}^T \rangle = \Lambda$;

- limitandoci alle prime k componenti principali abbiamo la *miglior* approssimazione dei dati in uno spazio a k dimensioni (dove per migliore intendiamo per esempio nel senso della norma di Frobenius o nel senso della somma delle distanze al quadrato fra i punti iniziali e le loro proiezioni nello spazio a k dimensioni). In altre parole volendo approssimare i nostri dati in un sottospazio a k dimensioni le prime k componenti principali $y_{i\nu} = \vec{e}_i^T \vec{x}_\nu$ e le rispettive "direzioni" \vec{e}_i sono quelle che rendono minima la quantità⁴⁴ $\sum_{\nu=1}^m (\vec{x}_\nu - \sum_{i=1}^k y_{i\nu} \vec{e}_i)^2$;
- di conseguenza se teniamo solo le k componenti principali si usa dire che spieghiamo una frazione

$$\frac{\sum_{i=1}^k \lambda_i}{\sum_{i=1}^n \lambda_i}$$

della varianza totale dei dati. È molto frequente il caso in cui i dati stanno in un sottospazio di dimensionalità molto inferiore dello spazio originale per cui possiamo analizzare i dati in uno spazio di dimensione più piccola. Quanto più piccola è stabilito dal numero di autovalori significativamente diversi da 0;

- si può dimostrare che se i dati hanno distribuzione Gaussiana le componenti principali massimizzano il contenuto di *informazione*;
- nei casi pratici raramente si calcolano gli autovettori della matrice C dato che il calcolo può essere critico dal punto di vista numerico. È più comodo usare la tecnica, più moderna e più stabile numericamente, della Singular Value Decomposition [SVD](#) ma, dal punto di vista teorico, le cose non cambiano.

⁴⁴nel semplice caso di riduzione ad una sola dimensione si ricava

$$\sum_{\nu=1}^m (\vec{x}_\nu - y_{1\nu} \vec{e}_M)^2 = \sum_{\nu=1}^m \vec{x}_\nu^2 - 2y_{1\nu} \vec{x}_\nu^T \vec{e}_M + y_{1\nu}^2 \vec{e}_M^T \vec{e}_M = \sum_{\nu=1}^m \vec{x}_\nu^2 - m \langle y_1^2 \rangle$$

e dato che $\langle y_1^2 \rangle$ è massimo se ne ricava che la somma iniziale è minima

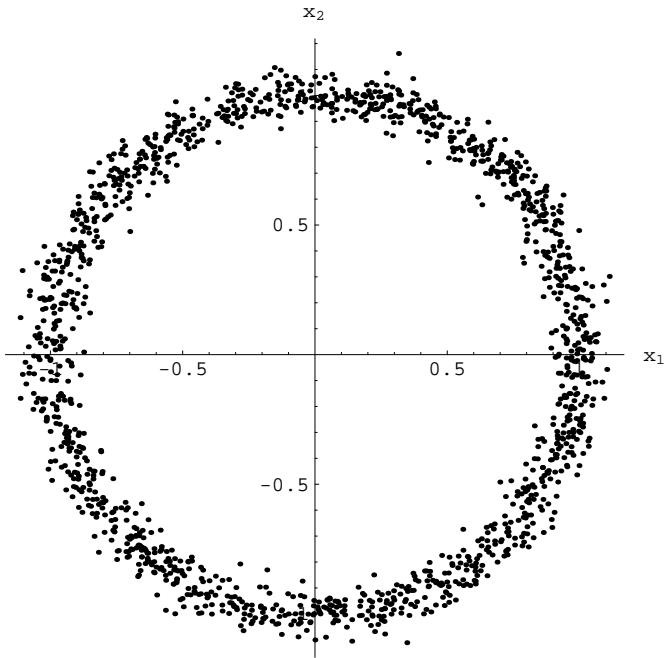


Figura 89: Una distribuzione bidimensionale di punti non adatta all’analisi delle componenti principali.

Concludiamo con i problemi di questa tecnica: il problema principale è che cambiando unità di misura delle variabili (per esempio lunghezze espresse in metri invece che in centimetri) *cambiano* le componenti principali. Questo si vede subito dato che passando da x_i a αx_i , pur rimanendo i dati a media nulla: $\langle x_i \rangle = \langle \alpha x_i \rangle = \langle y \rangle = 0$, in generale il cambiamento produce varianze diverse per le variabili trasformate y . Di conseguenza cambiano i valori sulla diagonale della matrice C e dunque gli autovalori cambiano a seconda delle unità di misura. Questo è un problema intrinseco del metodo, che può magari trovare una cura sintomatica nel considerare le variabili standardizzate $\frac{x_i}{\sigma_i}$, ma che non può essere curato alla radice.

Un altro problema è che, per certe distribuzioni di dati, le componenti principali non portano nessun miglioramento: si consideri per esempio il caso di figura 89 dove ci sono 1500 dati bidimensionali distribuiti uniformemente lungo un cerchio centrato nell’origine. In questo caso non ci sono direzioni che massimizzino la varianza (autovalori degeneri) ma invece una trasformazione non lineare dei dati, come per esempio passare a coordinate polari, potrebbe essere molto più significativa. Questo esempio mostra chiaramente anche l’effetto della scelta delle unità di misura: cambiando scala a uno degli assi il cerchio diventa un elisse, gli autovalori non sono più degeneri etc. etc. Per questi ed altri casi sono stati studiati altri metodi proiettivi che però noi non trattiamo.

Messaggio	Codifica 1	Codifica 2	Probabilità
Sole	00	0	$\frac{1}{2}$
Coperto	01	10	$\frac{1}{4}$
Pioggia	10	110	$\frac{1}{8}$
Neve	11	1110	$\frac{1}{8}$

Tabella 3: Tempo metereologico, due possibili codifiche e probabilità relative.

A.5 Cenni di Teoria dell'Informazione

Questa parte è tratta dai libri di Abramson [1] e Cover e Thomas [4].

Supponiamo di voler comprimere una certa file: qual è il massimo limite che possiamo raggiungere? Per rispondere a questa domanda occorre introdurre una quantità S nota come *entropia* o *informazione* già usata in termodinamica e scoperta nella sua nuova veste da [Claude Shannon](#) (figura 90) che negli anni del 1940 lavorava alla Bell Telephone. S ci fornirà un limite superiore alla massima compressione che si può effettuare sulla nostra file.

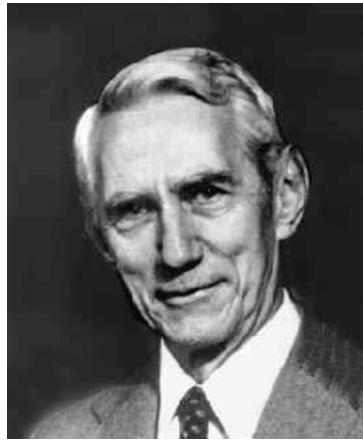


Figura 90: [Claude Shannon](#) (1916–2001) matematico e ingegnere elettronico

Procediamo con un esempio: supponiamo di voler trasmettere ad un amico lontano la situazione metereologica che riasumiamo in 4 possibilità esposte in tabella 3. Nel linguaggio della teoria dell'informazione diremo che noi siamo la *sorgente* di messaggi (o *simboli*, che indicheremo con x) che sono poi *codificati* per essere trasmessi all'amico. Per esempio nella seconda colonna della tabella 3 c'è una codifica che ha una lunghezza fissa di due bit. In questo caso è ovvio che, qualsiasi tempo faccia, la lunghezza del messaggio trasmesso sarà di 2 bit. Se siamo in una zona nella quale i 4 tipi di tempo metereologico sono equiprobabili non c'è nulla da dire. Se invece fossimo in una regione con delle probabilità metereologiche uguali a quelle in tabella 3 con la prima codifica la lunghezza media del messaggio sarebbe sempre 2 bit ma calcoliamo quanto verrebbe con la seconda codifica, quella della terza colonna. Con ovvia notazione

$$E[l(x)] = \sum_{x=1}^4 p(x)l(x) = \frac{1}{2}1 + \frac{1}{4}2 + \frac{1}{8}3 + \frac{1}{8}4 = \frac{15}{8} = 1.875$$

che ci dice che abbiamo risparmiato il 6.25% nella lunghezza media del messaggio! È possibile fare di meglio? Ve lo lascio per esercizio ma prima

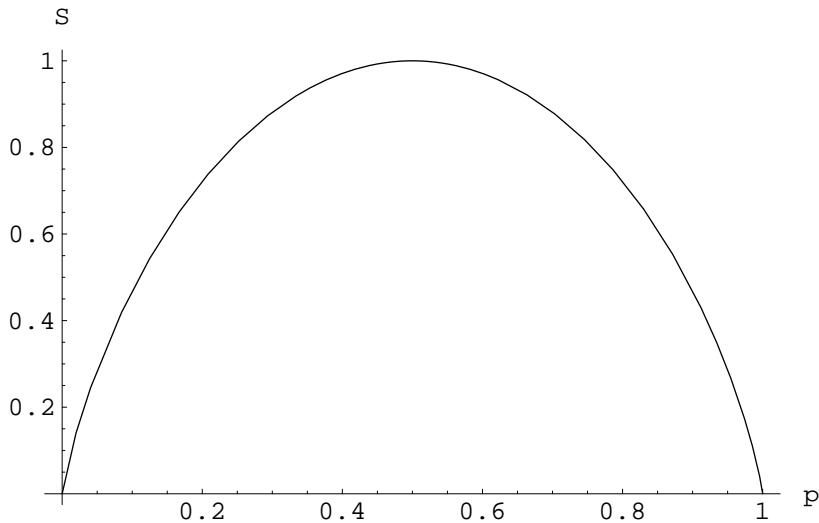


Figura 91: Informazione \mathcal{S} per una sorgente binaria al variare di p , la probabilità che venga emesso il primo simbolo (56).

sappiate che il codice deve essere univocamente decodificabile e leggete il prossimo capoverso.

Si può dimostrare che esiste un limite inferiore alla lunghezza media di un messaggio che è data dalla quantità, detta *entropia* o *informazione* della *sorgente*

$$\mathcal{S}(x) = \sum_x p(x) \log_2 \frac{1}{p(x)} \quad (55)$$

che dipende dalla distribuzione di *probabilità* dei *simboli* provenienti dalla sorgente (e *non* dai simboli stessi) e si misura in *bit*⁴⁵.

Facilmente calcoliamo che nel nostro esempio metereologico con eventi equiprobabili con $p(x) = 1/4$ l'informazione viene $\mathcal{S}(x) = 2$ bit dunque la prima codifica è ottimale e non esiste nessun'altra codifica che potrebbe dare una lunghezza media minore di 2 bit. Nel caso di eventi con le probabilità viste si trova che vale

$$\mathcal{S}(x) = \frac{1}{2} \log_2 2 + \frac{1}{4} \log_2 4 + 2 \frac{1}{8} \log_2 8 = \frac{7}{4} = 1.75$$

e dunque potrebbero esistere altre codifiche che permettono di produrre messaggi di lunghezza media minore di 1.875 ma possiamo già escludere che

⁴⁵Volendo essere più precisi data la variabile X distribuita con probabilità $p_X(x)$ l'entropia è il *funzionale*

$$\mathcal{S}(p_X) = \sum_{x \in X} p_X(x) \log_2 \frac{1}{p_X(x)} .$$

esista una codifica che dia lunghezza media minore di 1.75 bit. Approfondendo questo argomento si può vedere se e come si può costruire un codice ottimale ma noi tralasciamo questo argomento e andiamo avanti.

Supponiamo di avere a che fare con una sorgente che può produrre solo due simboli, necessariamente le rispettive probabilità saranno p e $1 - p$ di conseguenza l'entropia (55) di questa sorgente è

$$\mathcal{S}(x) = p \log_2 \frac{1}{p} + (1 - p) \log_2 \frac{1}{1 - p} \quad (56)$$

e possiamo studiare come varia \mathcal{S} al variare di p : il grafico è riportato in figura 91. Notiamo che la funzione è simmetrica rispetto al punto $p = 1/2$ nel quale assume il valore massimo $\mathcal{S} = 1$ bit e che assumiamo valere 0 per $p = 0, 1$ dato che $\lim_{p \rightarrow 0} p \log_2 \frac{1}{p} = 0$. Si può riassumere a parole dicendo che la sorgente da informazione nulla quando si sa con certezza il messaggio che produce ed è massima, un bit, quando invece l'incertezza sul messaggio è massima.

Possiamo definire l'informazione $I(x)$ contenuta nel simbolo x come

$$I(x) = \log_2 \frac{1}{p(x)} \quad (57)$$

per cui l'informazione contenuta nel messaggio sole è $\log_2 2 = 1$ bit mentre quella contenuta in negli altri messaggi è rispettivamente $\log_2 4 = 2$ e $\log_2 8 = 3$ bit.

Con questa nuova definizione di informazione di un messaggio (o simbolo) possiamo interpretare l'informazione di una sorgente (55)

$$\mathcal{S}(x) = \sum_x p(x)I(x) = E[I(x)] \quad (58)$$

e cioè come la *quantità media di informazione* per un simbolo della sorgente.

Qualche altro esempio: una moneta equa è una sorgente di entropia $\mathcal{S}(x) = 1$ bit. Quanta informazione riceviamo invece quando sentiamo una parola della lingua italiana? Se supponiamo rozzamente che vi siano 10^4 parole diverse e che il nostro interlocutore scelga ogni volta una parola a caso (e per certe persone questa approssimazione potrebbe sembrare plausibile) riceviamo $\log_2 10^4 \simeq 13.3$ bit. E se sentiamo due parole? Se supponiamo che siano indipendenti fra loro riceveremo esattamente il doppio di informazione 26.6 bit. Infatti se le parole sono indipendenti la probabilità di avere due parole x e y vale $p(x, y) = p(x)p(y)$ e di conseguenza $I(x, y) = I(x) + I(y)$. Dunque per come è definita l'informazione ha la desiderabile proprietà di essere proporzionale alla lunghezza del messaggio (se non vi sono correlazioni).

Infine proviamo a calcolare quanta informazione c'è in un'immagine della televisione ad alta definizione: essa è formata da $1920 \times 1080 \simeq 2.1 \cdot 10^6$

pixels ognuno dei quali può assumere uno dei $256^3 \simeq 1.7 \cdot 10^7$ colori riproducibili da uno schermo TV HD. Dunque in totale vi sono $(256^3)^{1920 \cdot 1080} = 2^{8 \cdot 3 \cdot 1920 \cdot 1080} \simeq 10^{1.5 \cdot 10^7}$ immagini diverse e, se supponiamo che siano tutte equiprobabili (e questa è l'ipotesi meno realistica), l'informazione contenuta in una singola immagine è $\log_2 2^{8 \cdot 3 \cdot 1920 \cdot 1080} \simeq 5 \cdot 10^7$ bit il che prova che il detto che recita: “un'immagine vale più di 1.000 parole” sarebbe più preciso nella forma: “un'immagine TV HD vale più di 3.741.835 di parole”!⁴⁶

Generalizziamo quanto già visto per il caso di una sorgente con due messaggi e cioè che l'entropia della sorgente (o se si preferisce la quantità media di informazione per simbolo) è massima quando i simboli sono equiprobabili.

Teorema 3 *Per una sorgente con W simboli si ha*

$$\mathcal{S}(x) \leq \log_2 W$$

e l'uguaglianza vale se e solo se tutti i simboli sono equiprobabili con probabilità $p = 1/W$.

Dimostrazione Usando la (55) e la relazione $\sum_{x=1}^W p(x) = 1$ calcoliamo

$$\begin{aligned} \mathcal{S}(x) - \log_2 W &= \sum_{x=1}^W p(x) \log_2 \frac{1}{p(x)} - \log_2 W = \\ &= \sum_{x=1}^W p(x) \log_2 \frac{1}{p(x)} - \sum_{x=1}^W p(x) \log_2 W = \\ &= \sum_{x=1}^W p(x) \log_2 \frac{1}{Wp(x)} = \log_2 e \sum_{x=1}^W p(x) \log \frac{1}{Wp(x)} \end{aligned}$$

dove abbiamo cambiato base al logaritmo (con la: $\log_2 x = \log_2 e \log x$) per poter usare la relazione

$$\log x \leq x - 1$$

dove l'uguaglianza vale solo per $x = 1$. Applicando questa diseguaglianza al nostro caso da

$$\begin{aligned} \mathcal{S}(x) - \log_2 W &= \log_2 e \sum_{x=1}^W p(x) \log \frac{1}{Wp(x)} \leq \\ &\leq \log_2 e \sum_{x=1}^W p(x) \left[\frac{1}{Wp(x)} - 1 \right] = \\ &= \log_2 e \left(\frac{W}{W} - 1 \right) = 0 \end{aligned}$$

⁴⁶mentre invece per una vecchia TV con $720 \times 576 \simeq 4.15 \cdot 10^5$ pixels: “un'immagine TV vale più di 749.058 parole”

la quale prova la prima parte del teorema. Per la seconda parte ritorniamo alla diseguaglianza sul logaritmo e osserviamo che l'uguaglianza vale se, e solo se, $Wp(x) = 1$ per ogni x il che significa che i simboli sono tutti equiprobabili con probabilità $p = 1/W$. \square

Alla luce di questo risultato notiamo che i valori che abbiamo calcolato per l'informazione contenuta in una parola di italiano o in un'immagine televisiva sono solo *limiti superiori* alle quantità vere dato che per calcolarle abbiamo supposto tutti i simboli equiprobabili.

Infine un'applicazione della *legge dei grandi numeri* che mostra un'altra proprietà dell'entropia: secondo questa legge date n variabili indipendenti e identicamente distribuite (i.i.d. variables) al crescere di n si ha convergenza (in probabilità)

$$\frac{1}{n} \sum_{i=1}^n x_i \xrightarrow{\text{P}} E[x]$$

per esempio la media campionaria del numero di teste in n lanci di una moneta tende al valore di aspettazione del numero di teste. Questo permette di dimostrare il Teorema dell'“Asymptotic Equipartition Property” (AEP)

Teorema 4 *dati n messaggi i.i.d. la media campionaria dell'informazione di un messaggio tende (in probabilità) all'entropia della sorgente e cioè*

$$\frac{1}{n} \sum_{i=1}^n \log \frac{1}{p(x_i)} \xrightarrow{\text{P}} E[\log \frac{1}{p(x)}] = \mathcal{S}(x) .$$

Dato che $\sum_{i=1}^n \log \frac{1}{p(x_i)} = \log \frac{1}{p(x_1, x_2, \dots, x_n)}$ possiamo dunque approssimare

$$p(x_1, x_2, \dots, x_n) \simeq 2^{-n\mathcal{S}(x)}$$

e potremo definire le *sequenze tipiche* x_1, x_2, \dots, x_n come quelle che hanno questa probabilità che può essere calcolata quando sia nota l'entropia $\mathcal{S}(x)$. Questo permette di dividere tutte le possibili sequenze x_1, x_2, \dots, x_n in due classi: quelle tipiche e le altre; si può anche dimostrare che le sequenze tipiche sono approssimativamente $2^{n\mathcal{S}(x)}$ (nel caso di eventi equiprobabili le sequenze tipiche sono tutte le sequenze possibili, infatti $2^{n\mathcal{S}(x)} = 2^{n \log W} = W^n$ ma le cose sono molto diverse nei casi meno banali). Questa proprietà può essere riassunta dicendo “quasi tutti gli eventi sono quasi egualmente sorprendenti”.

Riassumendo abbiamo introdotto una quantità, l'entropia di una sorgente $\mathcal{S}(x)$, che:

- è un funzionale della distribuzione di probabilità dei messaggi della sorgente (e *non dipende* dai singoli messaggi);
- è il valor medio dell'informazione di un messaggio, ovverosia una misura dell'incertezza media su un messaggio;

- per una sorgente con W messaggi è limitata da $0 \leq \mathcal{S}(x) \leq \log_2 W$ (per esercizio dimostrate la prima diseguaglianza e verificate per quali distribuzioni vale $\mathcal{S}(x) = 0$);
- nel caso di simboli equiprobabili $2^{\mathcal{S}(x)} = W$ e, nel caso generico, si generalizza la quantità $2^{\mathcal{S}(x)}$ a stima del numero di simboli disponibili;
- permette di calcolare la probabilità e il numero delle sequenze tipiche.

Concludiamo questa parte con un cenno al caso in cui ci sia una variabile continua x con densità di probabilità $p(x)$, in questo caso l'*entropia differenziale* è definita, con ovvia generalizzazione,

$$s(x) = \int p(x) \log_2 \frac{1}{p(x)} dx \quad (59)$$

e, per esempio, per una variabile con distribuzione uniforme nell'intervallo $[0, a]$ si ha $p(x) = 1/a$ e facilmente si trova

$$s(x) = \int_0^a \frac{1}{a} \log_2 a dx = \log_2 a$$

che ricorda il caso discreto dato che in quel caso, per una distribuzione uniforme, $\mathcal{S}(x) = \log_2 W$, mentre qui $s(x) = \log_2 a$ e a rappresenta una “misura” di tutti gli x accessibili; tuttavia in questo caso, se $a < 1$, l'entropia differenziale può risultare negativa ma, in ogni caso, $2^{s(x)} = a$ (come esercizio provate a verificare che l'entropia differenziale della Gaussiana vale $1/2 \log_2 2\pi e \sigma^2$).

Possiamo intuirne la ragione confrontandola con l'entropia della variabile discretizzata x_i con la quale si intende che a questa variabile associamo l'intervallo centrato in x_i e di ampiezza Δ al quale è associata la probabilità discreta $q(x_i)$

$$\int_{x_i - \frac{\Delta}{2}}^{x_i + \frac{\Delta}{2}} p(x) dx \simeq p(x_i) \Delta := q(x_i)$$

come potrebbe essere, per esempio, una variabile continua rappresentata con un numero floating point di n bit e in questo caso avremmo $\Delta = 2^{-n}$. Possiamo calcolare l'entropia per la variabile discretizzata con

$$\mathcal{S}(x_i) = \sum_i q(x_i) \log_2 \frac{1}{q(x_i)} = \sum_i p(x_i) \Delta \log_2 \frac{1}{p(x_i) \Delta} = \sum_i p(x_i) \Delta \log_2 \frac{1}{p(x_i)} - \log_2 \Delta$$

e visto che al tendere $\Delta \rightarrow 0$ il primo termine tende all'entropia differenziale $s(x)$ ricaviamo la relazione fra le due entropie

$$\mathcal{S}(x_i) + \log_2 \Delta \rightarrow s(x)$$

per esempio la versione quantizzata a n bit di una variabile continua ha entropia che vale approssimativamente $\mathcal{S}(x_i) - n \simeq s(x)$ che mostra come può nascere un entropia differenziale negativa per una variabile continua.

Anche per una variabile continua si può dimostrare che l'entropia differenziale è massima quando la distribuzione è uniforme $p(x) = 1/a$.

★ ★ *

Per procedere nella direzione che ci interessa analizziamo cosa succede se la nostra sorgente emette un simbolo x che, attraverso un canale di comunicazione, viene trasmesso e ricevuto come y . Esempi potrebbero essere l'input e l'output di una rete neurale oppure il messaggio trasmesso da un satellite e ricevuto a terra. Se il canale è perfetto e senza rumore (noiseless channel) varrà $y = x$ e tutta l'informazione della sorgente sarà ritrovata all'altro capo del canale. Naturalmente questo caso non è interessante e noi studiamo il caso generale in cui c'è una sorgente che emette messaggi x con distribuzione $p(x)$ e dopo il canale mi ritrovo con messaggi y con distribuzione $p(y)$. A questo punto posso anche pensare di avere due sorgenti x e y e ci chiederemo quanta sia l'informazione di x reperibile in y .

Cominciamo da una cosa piuttosto tecnica e definiamo la distanza di Kullback Leibler (o entropia relativa) fra due distribuzioni $p(x)$ e $q(x)$ come

$$D_{KL}(p(x), q(x)) := \sum_x p(x) \log_2 \frac{p(x)}{q(x)} \quad (60)$$

che, nonostante il nome, non è una distanza dato che non è simmetrica ($D_{KL}(p(x), q(x)) \neq D_{KL}(q(x), p(x))$) e non vale la proprietà triangolare. Vale però

$$D_{KL}(p(x), q(x)) \geq 0$$

l'uguaglianza valendo se e solo se $p(x) = q(x)$ il che si dimostra facilmente cambiando base al logaritmo ed usando la relazione $\log \frac{1}{x} \geq 1 - x$.

Ritorniamo al nostro problema e osserviamo che dal punto di vista matematico possiamo considerare la distribuzione multivariata⁴⁷ $p(x, y)$ di x e

⁴⁷Date due variabili x e y la loro distribuzione congiunta $p(x, y)$ si chiama multivariata, in questo caso a due dimensioni. Esempi banali sono il lancio di due monete o due dadi. Per il lancio di due monete eque è facile costruire la matrice 2×2 che rappresenta le 4 possibilità e scoprire che sono tutte uguali e valgono 1/4.

La probabilità di avere un certo valore di x per tutti gli y possibili è la cosiddetta probabilità marginale e si calcola con $p(x) = \sum_y p(x, y)$, analogamente $p(y) = \sum_x p(x, y)$. Per esempio per il lancio di due monete si vede che le probabilità marginali del primo lancio sono $p(\text{testa}) = p(\text{croce}) = 1/2$.

Nel semplice esempio del lancio di due monete abbiamo visto che $p(x, y) = p(x)p(y)$, si dice che la distribuzione $p(x, y)$ è *fattorizzata* e le variabili sono *indipendenti* il che significa che la conoscenza del risultato del lancio della prima moneta non mi dà nessuna informazione sul risultato del lancio della seconda moneta. Un altro esempio di distribuzione fattorizzata è la Gaussiana bidimensionale $p(x, y) \propto e^{-(x^2+y^2)/2} = e^{-x^2/2} e^{-y^2/2}$: anche qui qualsiasi sia il valore di x la distribuzione di y resta una Gaussiana unidimensionale con lo stesso valore medio e la stessa varianza. Facilmente si dimostra che se la distribuzione

y e, con ovvia estensione, definiamo l'entropia per questa distribuzione come

$$\mathcal{S}(x, y) = \sum_x \sum_y p(x, y) \log_2 \frac{1}{p(x, y)}$$

e per iniziare consideriamo il caso banale $y = x$, allora la somma sulle y contiene il solo termine $y = x$ e $p(x, y) = p(x, x) = p(x)$ e $\mathcal{S}(x, x) = \mathcal{S}(x)$. L'altro caso estremo (e sciagurato, perché in y non c'è traccia di x) è quello in cui le distribuzioni di x e y siano indipendenti fra di loro; e in questo caso

$$p(x, y) = p(x)p(y)$$

la distribuzione è fattorizzata e facilmente ricaviamo

$$\begin{aligned} \mathcal{S}(x, y) &= \sum_x \sum_y p(x)p(y) \log_2 \frac{1}{p(x)} \frac{1}{p(y)} = \\ &= \sum_x \sum_y p(x)p(y) \left(\log_2 \frac{1}{p(x)} + \log_2 \frac{1}{p(y)} \right) = \\ &= \sum_x \sum_y p(x)p(y) \log_2 \frac{1}{p(x)} + \sum_x \sum_y p(x)p(y) \log_2 \frac{1}{p(y)} = \\ &= \sum_y p(y) \sum_x p(x) \log_2 \frac{1}{p(x)} + \sum_x p(x) \sum_y p(y) \log_2 \frac{1}{p(y)} = \\ &= \mathcal{S}(x) + \mathcal{S}(y) \end{aligned}$$

e in questo caso l'informazione contenuta nella doppia sorgente è la somma delle informazioni delle sorgenti prese singolarmente.

È abbastanza intuitivo che questo è un limite superiore dato che se x e y sono correlate vi sarà meno informazione a disposizione; per un esempio

è fattorizzata la covarianza delle due variabili x e y è nulla (fatelo). Il viceversa non vale, ci sono alcuni esempi in figura 87.

Il caso di distribuzione fattorizzata rappresenta l'eccezione piuttosto che la norma: sono molto più frequenti i casi di distribuzioni multivariate nelle quali la conoscenza del valore x dà informazioni sulla y . Per restare in un semplice esempio si pensi al caso, simile a quello del lancio di due monete visto prima, dell'estrazione di due palline colorate, rispettivamente in bianco e nero, da un'urna. In questo caso se si estrae per prima la pallina bianca siamo sicuri che la seconda pallina sarà nera e viceversa: costruite anche in questo caso la matrice 2×2 che rappresenta la $p(x, y)$ e notate che le due distribuzioni marginali $p(x)$ e $p(y)$ sono uguali a quelle del lancio di due monete. In questo caso però la distribuzione non è fattorizzabile e in generale si può scrivere

$$p(x, y) = p(y|x)p(x) = p(x|y)p(y)$$

dove $p(y|x)$ è la *probabilità condizionale* di osservare y quando il primo risultato è stato x . Un altro esempio di distribuzione non fattorizzata è quella delle coppie di lettere nell'inferno di Dante riprodotta in figura 92. In questo caso le distribuzioni marginali sono entrambe la distribuzione delle singole lettere e le distribuzioni condizionali sono le distribuzioni della seconda lettera avendo fissato la prima lettera (o viceversa).

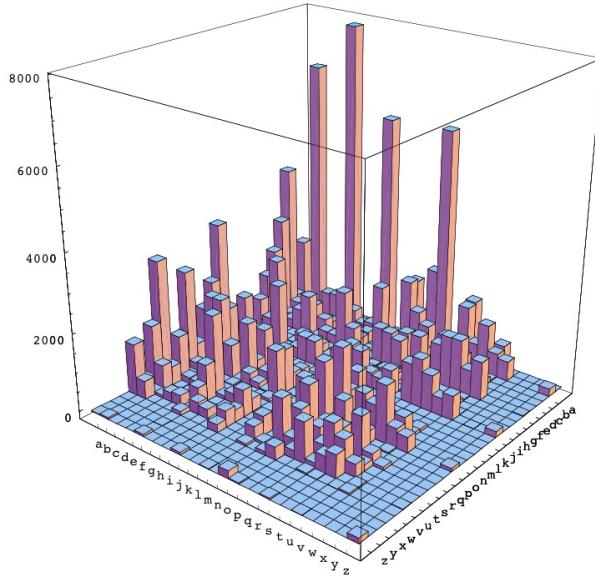


Figura 92: Esempio di distribuzione $p(x, y)$ con variabili correlate: frequenza delle coppie di lettere nell’Inferno di Dante (prima lettera a sinistra). La correlazione in una lingua se da un lato diminuisce la quantità di informazione trasmessa dall’altro aumenta la ridondanza permettendoci di capire messaggi anche se una parte è nascosta dal rumore.

di distribuzione $p(x, y)$ con variabili correlate si veda la figura 92. Infatti la relazione si può generalizzare a

$$\mathcal{S}(x, y) \leq \mathcal{S}(x) + \mathcal{S}(y) \quad (61)$$

l’uguaglianza valendo se e solo se le distribuzioni sono indipendenti e cioè appunto il caso $p(x, y) = p(x)p(y)$. Per dimostrare questa relazione definiamo come *mutua informazione* $M(x, y)$ fra x e y proprio la differenza fra queste quantità e cioè

$$M(x, y) := \mathcal{S}(x) + \mathcal{S}(y) - \mathcal{S}(x, y)$$

e facilmente si vede (fatelo, basta usare la definizione di distribuzione marginale $p(x) = \sum_y p(x, y)$) che la mutua informazione $M(x, y)$ altro non è che la distanza di Kullback Leibler fra le due distribuzioni $p(x, y)$ e $p(x)p(y)$ cioè

$$M(x, y) = \sum_x \sum_y p(x, y) \log_2 \frac{p(x, y)}{p(x)p(y)}$$

e dunque, per le proprietà della distanza di Kullback Leibler, resta dimostrata la (61). Abbiamo già visto che nel caso che le variabili siano indipendenti si ha $M(x, y) = 0$, esaminiamo ora il caso opposto delle variabili uguali e

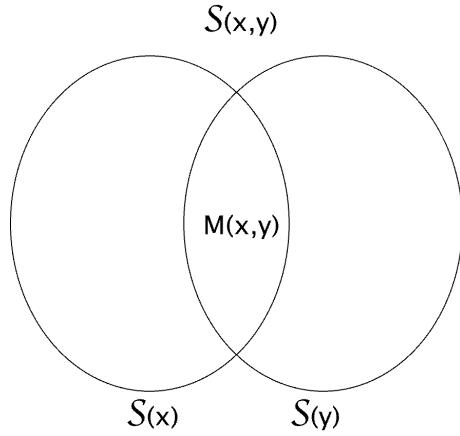


Figura 93: Rappresentazione grafica della mutua informazione: i due insiemi rappresentano rispettivamente $\mathcal{S}(x)$ e $\mathcal{S}(y)$, la loro unione rappresenta $\mathcal{S}(x,y)$ e la loro intersezione $M(x,y)$.

cioè $y = x$, in questo caso

$$M(x, y) = M(x, x) = \mathcal{S}(x) + \mathcal{S}(x) - \mathcal{S}(x, x) = \mathcal{S}(x)$$

che indica che il trasferimento di informazione fra x e y è totale. Possiamo riassumere le proprietà della mutua informazione con

$$\begin{aligned} M(x, y) &= \mathcal{S}(x) + \mathcal{S}(y) - \mathcal{S}(x, y) \geq 0 \\ M(x, x) &= \mathcal{S}(x) \end{aligned}$$

ed è più facile ricordare le proprietà della mutua informazione con un diagramma come quello in figura 93.

A.6 Ottimizzazione combinatoria e “Simulated Annealing”

Un applicazione di vasta portata della dinamica stocastica introdotta nel paragrafo 12.3 si ha nel 1983 quando Kirkpatrick, Gelatt e Vecchi propongono l'algoritmo di “[Simulated Annealing](#)” ispirato alla fisica statistica per la risoluzione di problemi di *ottimizzazione combinatoria*. In questi problemi si vuole minimizzare una funzione che dipende da variabili *discrete* per cui non sono applicabili gli usuali algoritmi di minimizzazione. Per esempio non si può applicare la discesa lungo il gradiente dato che non c’è uno spazio continuo per le variabili e il concetto stesso di direzione non è definito. L'esempio più famoso è il problema del commesso viaggiatore che deve trovare il percorso di lunghezza minima che lo porti a visitare n città senza passare due volte per la stessa città e ritornando al punto di partenza. Questo problema, che fa parte della famosa classe dei problemi [NP-completi](#), non ha in sostanza miglior soluzione che quella, brutale, di calcolare tutti gli $(n - 1)!$ percorsi diversi e scegliere poi quello di lunghezza minore. Visto come cresce $n!$ è chiaro che solo per valori di n piccoli il problema può essere risolto esattamente. In questo esempio la funzione da minimizzare è la lunghezza del percorso che dipende dal particolare ordine di visita delle città.

Gli autori notano che questi problemi assomigliano a quelli della ricerca dello stato fondamentale (stato a energia minima) in un cristallo nel quale, in molti casi, l'energia può dipendere da variabili discrete come lo spin. Se, per ipotesi, potessimo costruire un cristallo la cui energia H_i dell' i -esimo stato corrisponda numericamente, per restare nel nostro esempio, con la lunghezza dell' i -esimo cammino del commesso viaggiatore, allora potremmo trasformare la ricerca del cammino più breve nella ricerca dello stato fondamentale di questo ipotetico cristallo. In pratica potremmo raffreddare il cristallo e, al raggiungimento dello stato fondamentale, avremmo trovato anche il valore minimo della funzione di interesse.

In metallurgia è noto che in certi casi conviene abbassare molto lentamente la temperatura di una lega (ricottura o annealing) per permettere la redistribuzione degli atomi durante il raffreddamento e l'eliminazione dei difetti reticolari. In altre parole si ottiene il materiale migliore calando la temperatura così lentamente che il sistema resta sempre in equilibrio termico e dunque ad ogni temperatura occupa con maggior probabilità gli stati a energia minore dato che, per un sistema in equilibrio, la probabilità di trovarlo in un certo stato \vec{S} segue la distribuzione di Boltzmann (38)

$$P_{\vec{S}} = \frac{1}{Z} e^{-\frac{H_{\vec{S}}}{T}}$$

Nel 1953 Metropolis, Rosenbluth e Teller inventano una dinamica stocastica, cioè casuale, (presentata in generale nel paragrafo 12.3) per simulare un sistema a molti corpi. In particolare non si simula il sistema nel suo dettaglio fisico ma solamente le probabilità di transizione da uno stato ad

un altro data dalla (47) che riportiamo qui per comodità:

$$\mathcal{P}(\alpha \rightarrow \alpha') = \begin{cases} 1 & \text{se } \Delta H \leq 0 \\ e^{-\frac{\Delta H}{T}} & \text{altrimenti} \end{cases}$$

Il sistema simulato evolve e quando si trova in equilibrio la probabilità di trovarlo in un certo stato è data dalla distribuzione di Boltzmann e, per quanto riguarda le quantità misurabili macroscopiche, il sistema simulato è indistinguibile da un sistema fisico reale.

L'algoritmo di Metropolis è il seguente:

1. si genera *a caso* un cambio di configurazione e si calcola l'associata variazione di energia ΔH ;
2. se $\Delta H \leq 0$ si accetta la nuova configurazione;
3. se $\Delta H > 0$ si accetta la nuova configurazione con probabilità $P \propto e^{-\frac{\Delta H}{T}}$ (notiamo che configurazioni con $\Delta H > 0$ sono dunque possibili e più probabili quando la temperatura è alta).

Si dimostra che ripetendo questo algoritmo *abbastanza a lungo* il sistema andrà in equilibrio e la probabilità di trovare il sistema in un dato stato seguirà la distribuzione di Boltzmann (38).

Abbiamo ora tutti gli ingredienti per capire il funzionamento dell'algoritmo di Simulated Annealing (SA) o ricottura simulata. Dato un problema combinatorio si costruisce un sistema fisico virtuale nel quale i valori dell'energia corrispondono ai valori della funzione che vogliamo minimizzare e di questo sistema si cerca lo stato fondamentale *simulando* numericamente con l'algoritmo di Metropolis un lento raffreddamento fino allo zero assoluto. Per la distribuzione di Boltzmann a questa temperatura l'unico stato che ha probabilità non nulla è quello a energia minima, dunque se il raffreddamento è stato abbastanza lento alla fine il sistema si troverà nello stato fondamentale.

Riassumiamo il tutto in una serie di corrispondenze:

1. alle configurazioni del sistema fisico simulato corrispondono i valori delle variabili della funzione da minimizzare;
2. all'energia del sistema simulato corrisponde il valore della funzione da minimizzare;
3. alla temperatura corrisponde un parametro di controllo.

In pratica l'algoritmo è molto semplice da scrivere perché non è altro che una continua ripetizione dell'algoritmo di Metropolis con la contemporanea, lenta, diminuzione della temperatura del sistema. Il tutto deve essere ripetuto abbastanza a lungo da far sì che in ogni istante il sistema sia in

equilibrio termodinamico in modo che la distribuzione dei suoi stati seguia la distribuzione di Boltzmann. Mentre la simulazione procede si nota che ad alte temperature vengono fissate le caratteristiche grossolane del sistema e a basse temperature i dettagli.

Vista la semplicità di programmare l'algoritmo e la buona qualità delle soluzioni che è in grado di trovare (per esempio anche nel caso del commesso viaggiatore) l'algoritmo ha trovato grandissima diffusione per i problemi più vari.

Menzioniamo infine un risultato teorico più recente che prova che se la temperatura scende abbastanza lentamente (proporzionalmente all'inverso del logaritmo del numero di passi) l'algoritmo porta alla soluzione esatta con probabilità 1. In pratica un raffreddamento così lento risulta più lento della ricerca esaustiva della soluzione esatta ma fornisce all'algoritmo una patente di credibilità. Variando la rapidità con cui la temperatura scende si possono ottenere soluzioni approssimate di diversa qualità.

A.7 Qualche indirizzo di rete

Links interessanti (verificati dicembre 2015):

<https://nips.cc/Conferences/2007/Schedule?type=Tutorial> NIPS 2007 Tutorials (per esempio la visione T.Poggio ed altri)

<http://www-gap.dcs.st-and.ac.uk/~history/index.html> Un sito di biografie di matematici dal quale ho tratto alcune immagini di questa dispensa.

<http://www.mth.kcl.ac.uk/~iwilde/notes/nn/index.html> sito di I. F. Wilde (King's College - London) dove si trova la dispensa: Neural Networks

<http://grey.colorado.edu/CompCogNeuro/index.php/CCNBook/Main> Computational Explorations in Cognitive Neuroscience (O'Reilly & Munakata, 2000, free wiki textbook)

Links sulla didattica di questo corso:

Didattica di Marco Budinich: <http://wwwusers.ts.infn.it/~mbh/Teaching.html>

Sito Moodle del corso dell'anno accademico 2016–2017:

<https://moodle2.units.it/course/view.php?id=1457> (Moodle @ Uni-Ts: <http://moodle2.units.it>)

Bibliografia

- [1] N. Abramson. *Information Theory and Coding*. Electronic Sciences Series. McGraw-Hill, New York, p. xvi–201, 1963.
- [2] A. J. Bell and T. J. Sejnowski. An information-maximization approach to blind separation and blind deconvolution. *Neural Computation*, 7(6):1129–1159, Nov 1995.
- [3] E. Benedetti and M. Budinich. Neural relax. *Neural Computation*, 24(11):3091–3110, November 2012.
- [4] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley Interscience. John Wiley & Sons Inc., New York, second edition, p. xxiv–748, 2006. Biblioteca DF: I–431, personal copy.
- [5] V. Dotsenko. *An Introduction to the Theory of Spin Glasses and Neural Networks*, volume 54 of *World Scientific Lecture Notes in Physics*. World Scientific, Singapore, p. viii–156, 1994. Biblioteca DF: I–387.
- [6] J. S. Griffith. *Mathematical Neurobiology*. Academic Press, London, p. x–162, 1971. Biblioteca DF: I–224.
- [7] S. Haykin. *Neural Networks — A Comprehensive Foundation*. Prentice Hall, second edition, p. xxii–842, 1999. Biblioteca Dipartimento di Energetica: 08/02 71.
- [8] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley, Santa Fe Institute, p. xxii–328, 1991. Biblioteca DF: I–295, personal copy.
- [9] S. Laughlin. A Simple Coding Procedure Enhances a Neuron’s Information Capacity. *Zeitschrift für Naturforschung*, 36:910–912, 1981.
- [10] B. Müller and J. Reinhardt. *Neural Networks — An Introduction*. Springer Verlag, Berlin Heidelberg, p. xiv–266, 1991. Biblioteca DF: I–296.
- [11] J. Nadal and N. Parga. Information processing by a perceptron in an unsupervised learning task. *Network: Computation in Neural Systems*, 4:295–312, 1993.
- [12] S. E. Palmer. *Vision science: photons to phenomenology*. The MIT Press, p. xxii–810, 2002. Biblioteca Psicología: 01./2342.

Indice

1 Introduzione	2
2 Una breve introduzione biologica	2
3 Modelli matematici ispirati ai neuroni	10
4 Apprendimento per i neuroni di McCulloch e Pitts	14
4.1 Formalizzazione matematica e notazione	15
4.2 Altri tipi di apprendimento	19
4.3 La regola di Hebb	20
5 Perceptrons	21
5.1 E ora ?	27
5.2 Il Perceptron continuo	27
6 Le reti “feed-forward”	32
6.1 La “back-propagation” in pratica	37
6.2 Caratteristiche delle reti “feed-forward”	39
7 La capacità del Perceptron	41
7.1 La capacità del Perceptron	44
7.2 Approssimazione Gaussiana di $C(m, n)$	45
7.3 La dimensione di Vapnik e Cervonenkis d_{VC}	45
8 Il problema della generalizzazione	46
9 Apprendimento non supervisionato	54
9.1 Un esempio iniziale	54
9.2 La regola di Oja	56
9.3 La regola di Sanger	58
9.4 Le reti con l'apprendimento competitivo	61
9.5 Le reti di Kohonen	65
10 Reti neurali e teoria dell'informazione	77
10.1 Un semplice esempio biologico: l'occhio della mosca	77
10.2 Un altro algoritmo di apprendimento per il Perceptron	79
10.3 Uno strato di h Perceptron	80
10.4 Algoritmi di apprendimento basati sulla teoria dell'informazione	84
11 Memorie associative	88
11.1 Il cammino casuale unidimensionale	93
11.2 Riprendiamo il conto sospeso	95
11.3 Osservazioni sparse sulla matrice dei pesi	98
11.3.1 Efficienza della memoria associativa	103

11.3.2 Suggerimenti per simulare una memoria associativa	104
11.4 Uso della funzione Energia nelle memorie associative	105
12 Meccanica statistica e reti neurali	108
12.1 Il modello di Ising	109
12.2 Semplici esempi fisici	112
12.3 Cenni alla dinamica stocastica	115
12.4 Modello di Ising e memorie associative	118
12.5 Soluzioni del modello di Hopfield nel caso $\alpha \rightarrow 0$	119
12.6 Il caso $\alpha \neq 0$	123
12.7 La macchina di Boltzmann	124
12.8 Ancora la regola di Hebb	130
13 Una breve introduzione alla visione	131
13.1 Cenni alla visione umana	132
13.1.1 La retina	132
13.1.2 Il nucleo genicolato laterale LGN	135
13.1.3 La corteccia striata	137
13.2 La teoria della frequenza spaziale	140
13.3 Approccio computazionale: il riconoscimento dei bordi	144
13.3.1 Calcolo numerico delle derivate	148
13.4 Problemi mal posti	149
13.4.1 Definizione dei problemi ben posti	149
13.4.2 Un esempio unidimensionale	150
13.5 Altre interpretazioni della visione	151
13.5.1 Interpretazioni basate sulle reti neurali	153
13.5.2 Interpretazioni basate sulla distribuzione delle immagini naturali	153
13.5.3 Ipotesi dei canali separati	155
13.6 Il “Cappello Messicano”	155
14 Conclusioni	156
A Appendici	157
A.1 Metodi di minimizzazione	157
A.2 La distribuzione binomiale	159
A.3 La matrice di covarianza	161
A.4 Il metodo delle componenti principali (PCA)	163
A.5 Cenni di Teoria dell'Informazione	168
A.6 Ottimizzazione combinatoria e “Simulated Annealing”	178
A.7 Qualche indirizzo di rete	180
Bibliografia	181
Indice	182