Capitolo 1

Introduzione

Questo lavoro di tesi affronta il problema del riconoscimento, eseguito da un algoritmo implementato al computer, di tracce lasciate da particelle cariche, emesse in un evento di collisione nucleo-nucleo ad energie ultra-relativistiche, e rivelate dal multi-rivelatore ALICE [ALICE-TP]. L'approccio adottato è stato quello di fare uso delle cosiddette "memorie auto-associative di Hopfield" [Free91, Kot93, Hum90] che rientrano nella vasta famiglia delle reti neurali artificiali (RNA). Lo scopo di questo capitolo è quello di introdurre il lettore alla problematica che sarà oggetto del presente studio, fornendogli un'idea sia qualitativa che quantitativa dei problemi di cui è lastricato il cammino che conduce al conseguimento dei risultati che ne costituiscono l'obiettivo. Nei prossimi paragrafi sarà dunque fornita una descrizione del multi-rivelatore ALICE ed una panoramica del lavoro di tesi.

1.1 ALICE

Il multi-rivelatore ALICE (A Large Ion Collider Experiment) [ALICE-TP], in fase attuale di progettazione e realizzazione presso il CERN di Ginevra, è uno dei quattro grandi rivelatori, di cui è già stata approvata la costruzione, che saranno installati presso l'acceleratore Large Hadron Collider (LHC) che è in costruzione nello stesso laboratorio internazionale. Da un punto di vista tecnico, LHC è un collisionatore, del diametro di circa 27 km, che accelererà protoni e ioni Ca e Pb ad un'energia di circa 3 TeV per nucleone per ciascuno dei due fasci e con luminosità che andranno da qualche unità in $10^{-27}\,\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$, nel caso degli ioni pesanti, a $10^{-30}\,\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$, nel caso dei protoni. La frequenza con cui si verificheranno le collisioni sarà di circa 8000 Hz, nel caso degli ioni pesanti, e 100 kHz, nel caso dei protoni. Tutto ciò fa sí che, quando LHC entrerà in funzione, nel 2005, l'energia disponibile nel centro di massa delle reazioni che avverranno nei suoi punti di interazione sarà quasi due ordini di grandezza più elevata di quella del più potente acceleratore oggi operante nel mondo, addirittura più elevata di quella che si raggiunge in alcune collisioni nucleo-nucleo indotte dai raggi cosmici.

ALICE, dei quattro multi-rivelatori di LHC, è quello che più di ogni altro è stato progettato per lo studio dei prodotti di collisioni ione-ione, in particolare Pb-Pb. Lo scopo della fisica degli ioni pesanti alle energie ultra-relativistiche è lo studio della materia fortemente interagente a valori estremi della temperatura e della densità di energia. La Cromo Dinamica Quantistica (QCD) predice che, a valori della temperatura e della densità di energia sufficientemente elevati, si innesca una transizione di fase che trasforma la materia adronica ordinaria in un plasma di quark e gluoni liberi (QGP). L'esistenza di tale

transizione, se confermata in laboratorio da esperimenti quali, appunto, ALICE, è di importanza fondamentale per le implicazioni che comporterebbe, in relazione al fatto che si ritiene sia avvenuta nell'Universo solo qualche decina di microsecondi dopo il Big Bang. Per di più, si crede che il suo studio possa giocare un ruolo molto importante allo scopo di comprendere appieno la struttura del "core" delle stelle di neutroni collassate. Le problematiche teoriche e fenomenologiche che stanno alla base dell'esperimento ALICE esulano, purtroppo, per la loro vastità e complessità, dallo scopo del presente lavoro di tesi per cui si rimanda il lettore interessato all'argomento alla letteratura scientifica relativa [Wong94].

In questa sede è di estrema importanza sottolineare che, siccome la temperatura e la densità della zona di interazione aumentano al diminuire del parametro d'urto della collisione, l'esperimento ALICE è dedicato espressamente alla rivelazione di quelli che sono solitamente indicati come "eventi centrali". È noto che in essi le particelle prodotte durante la collisione vengono emesse preferenzialmente in direzione trasversale all'asse di collisione. Per questo, i rivelatori che costituiscono ALICE sono stati progettati per coprire soprattutto la regione di spazio attorno a 90 gradi rispetto alla direzione dei fasci. Ciò spiega perché la regione sensibile centrale di tale grosso sistema di rivelatori sia compresa in un intervallo di pseudo-rapidità[1] tutto sommato piccolo attorno a pseudo-rapidità nulla. Si è progettato, infatti, di limitarsi alla regione $|\eta| \leq 0.9$, che corrisponde, geometricamente, ad una regione definita da coni di semiapertura di 45 gradi, con l'asse di rotazione diretto lungo l'asse del moto dei nuclei collidenti (vedi figura 1.1). Il sistema di riferimento adottato in ALICE è disposto in modo tale che l'asse z sia quello lungo il quale giace la direzione del moto dei proiettili, e gli assi x ed y siano disposti in modo da rispettare la regola della mano sinistra: precisamente, x punta verso il centro dell'anello dell'acceleratore LHC e y verso l'alto.

La distribuzione di molteplicità, in funzione della pseudo-rapidità, delle particelle emesse in una collisione centrale Pb-Pb all'energia di LHC è riportata in figura 1.2. Nell'intorno di pseudo-rapidità nulla (regione che viene indicata con il termine "mid-rapidity"), in cui si prevede il contributo maggiore all'emissione delle particelle provenienti dalla zona di interazione, e che quindi riveste particolare importanza per quanto detto sopra, la molteplicità totale ammonta a più di 12000 particelle per unità di pseudo-rapidità, di cui due terzi cariche. Se si tiene conto degli effetti elettromagnetici indotti dai fotoni, dei decadimenti e dei raggi δ dovuti alla ionizzazione prodotta dalle particelle cariche nei mezzi che costituiscono il rivelatore, si può comprendere l'enormità del numero di tracce con le quali si ha a che fare, del tutto nuovo in esperimenti di questo tipo, e l'arduità del compito che si intende assolvere, ossia quello della loro ricostruzione.

Alla elevata densità di tracce, si aggiunge l'evidenza che la distribuzione di impulso trasverso (perpendicolare alla direzione dei fasci incidenti) delle particelle emesse nella collisione (si veda la figura 1.3) presenta un massimo per bassi valori di questa grandezza, circa 200-250 MeV/c, per cui l'effetto del campo magnetico di 0.2 T (parallelo alla direzione dei fasci) in cui il multi-rivelatore è immerso, è quello di far intersecare le traiettorie delle particelle, introducendo un'ulteriore complicazione non banale all'assegnazione dei punti spaziali da esso ricostruiti alle tracce che li hanno generati. Per quanto sopra si capisce come, la progettazione del multi-rivelatore ALICE, cosí come l'analisi dei dati che da esso proverranno, stia rappresentando una sfida tecnologica senza precedenti.

Alcune "viste" del multi-rivelatore ALICE sono riportate nelle figure 1.5-1.4, dalle

[1] La pseudo-rapidità di una particella è definita come $\eta = -\ln \tan \frac{\theta}{2}$ dove θ è l'angolo polare di emissione calcolato rispetto alla direzione dei fasci collidenti.

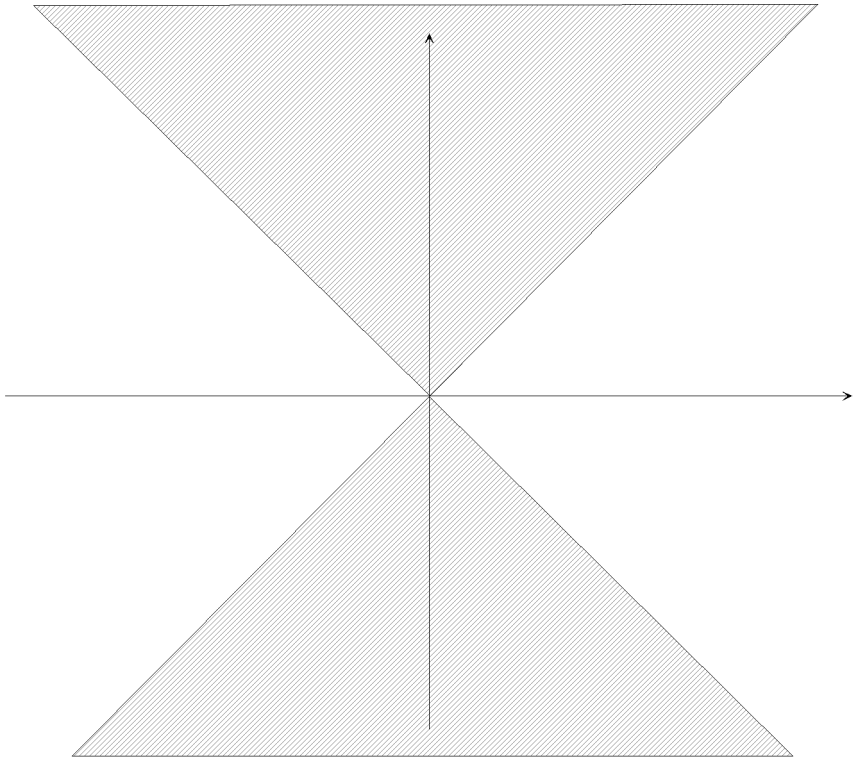

Figura 1.1: La regione segnata corrisponde alla zona a cui viene rivolta l'attenzione in modo particolare. Si immagini che il disegno abbia simmetria sferica attorno all'asse orizzontale, lungo il quale i nuclei collidono (z).

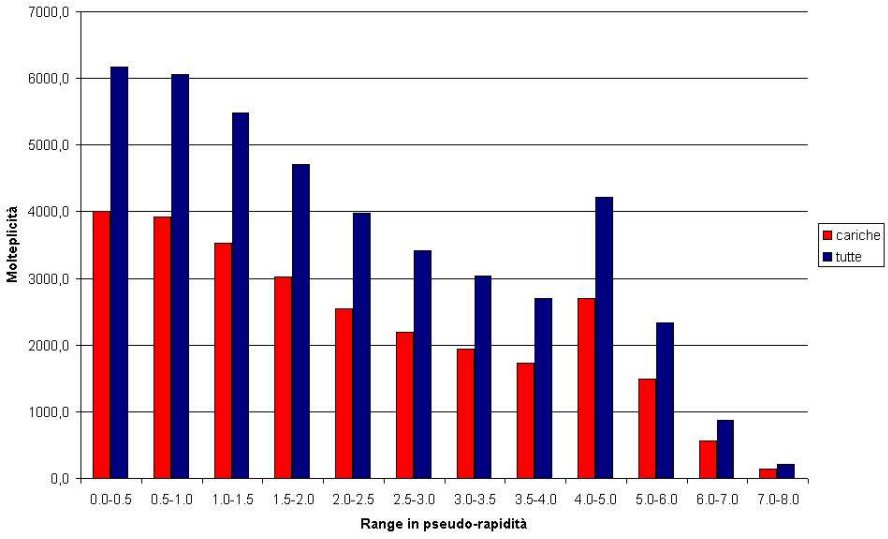

Figura 1.2: Distribuzione di molteplicità, in funzione della loro pseudo-rapidità, delle particelle emesse in una collisione centrale Pb-Pb all'energia di LHC. L'istogramma più scuro si riferisce a tutte le particelle, mentre quello più chiaro solamente a quelle cariche. Si noti che la non monotonicità della distribuzione è dovuta semplicemente al fatto che gli ultimi quattro bin degli istogrammi sono di larghezza doppia rispetto ai primi otto.

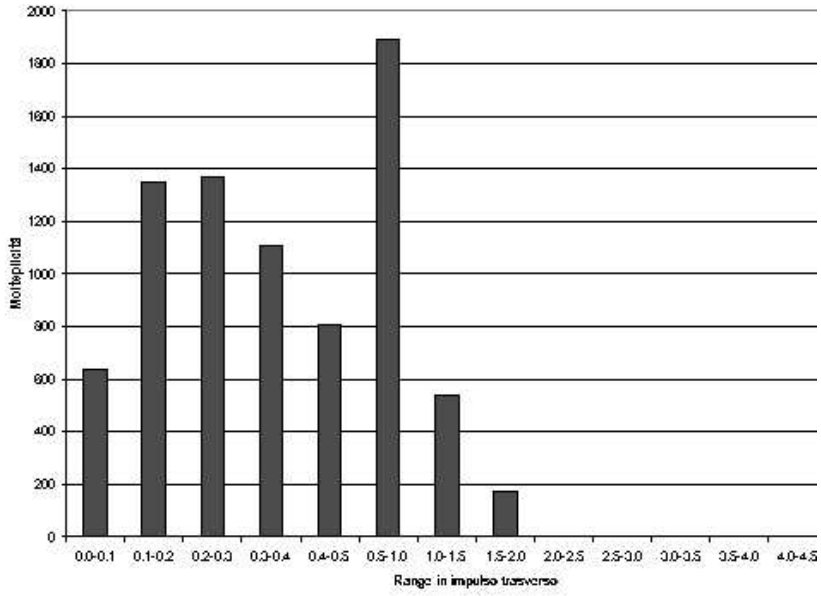

Figura 1.3: Distribuzione di impulso traverso delle particelle emesse in una collisione centrale Pb-Pb all'energia di LHC. L'impulso traverso è riportato in ascissa in unità di GeV/c. Si noti che la non monotonicità della distribuzione è dovuta semplicemente al fatto che gli ultimi tre bin dell'istogramma sono di larghezza quintupla rispetto ai primi cinque.

quali è pure possibile apprezzare le sue reali dimensioni. Se si osserva la sezione di ALICE in un piano che contiene la direzione dei fasci incidenti, è possibile distinguere una parte centrale, che copre, come si è detto, la regione di pseudorapidità $|\eta| \leq 0.9$, e che è installata all'interno del magnete avente sezione trasversale ottagonale, ed un braccio laterale, che copre la regione angolare polare compresa tra 2 gradi e 10 gradi.

La parte centrale comprende i seguenti rivelatori:

Inner Tracking System (ITS). È il rivelatore più interno del sistema, ed è costituito da sei strati grossolanamente cilindrici di micro-rivelatori di silicio. Il suo scopo è quello di circondare le immediate vicinanze del punto di interazione, per ricavare le caratteristiche delle particelle emesse nelle collisioni, permettendo una risoluzione elevata del parametro d'impatto e quindi una determinazione del vertice della interazione e di quelli secondari con un'alta precisione. Uno degli scopi più importanti dell'ITS è infatti quello di rilevare la presenza di particelle dotate di stranezza o incanto, localizzandone il punto di decadimento.

Time Projection Chamber (TPC). Prosegue il processo di rivelazione di punti che consente di ricostruire le tracce appena al di fuori dell'ITS. Essa è un rivelatore a gas che fornisce la maggior parte dei punti ottenuti per ogni traccia, e come tale è stata l'unico ad essere analizzato e sfruttato nello studio che seguirà. Per questo, maggior dettaglio sulle caratteristiche salienti della TPC, dal punto di vista del presente lavoro, sarà fornito nel capitolo 5.

Photon Spectrometer (PHOS). È disegnato per valutare la temperatura della zona di interazione rivelando i fotoni che ne emergono.

Particle Identification (PID). Questa sezione svolge il compito di riconoscere la massa,

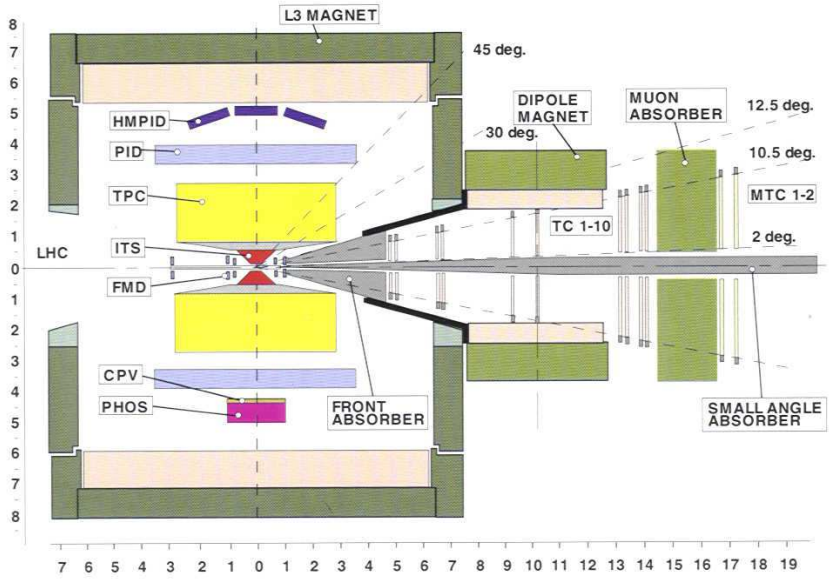

Figura 1.4: prospetto laterale del multirivelatore ALICE. Le scale laterale e inferiore, espresse in metri, consentono di stimare le dimensioni dell'apparato.

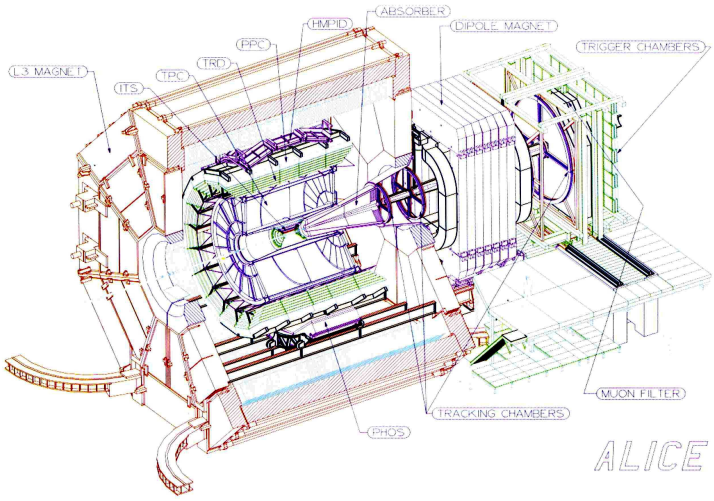

Figura 1.5: prospetto assonometrico dell'apparato. Si noti la presenza di un ulteriore rivelatore, il TRD, che nella precedente figura non compare in quanto la sua aggiunta è stata approvata di recente.

e quindi la natura, delle particelle rivelate. Se infatti le particelle di energia più bassa possono essere identificate dalla loro perdita di energia, per quelle ad alta energia tale procedimento viene eseguito misurandone il tempo di volo necessario a percorrere una distanza nota. Per energie ancor maggiori esiste il rivelatore HMPID che si basa sulla tecnologia RICH (Ring Image CHerenkov), che misura l'impulso ed il tipo delle particelle attraverso una misura dell'effetto Cherenkov indotto in un cristallo di CsI.

Il braccio laterale comprende un rivelatore per muoni, indicato come *Muon Arm*. Un suo scopo è quello di misurare le particelle J/Ψ e Υ, che decadono in coppie di muoni aventi cariche opposte.

1.2 Piano della tesi

1.2.1 Capitolo 2

Il capitolo 2 è dedicato a trattare i concetti generali che è bene avere chiari per lavorare con le reti neurali. Tali concetti non sono quelli relativi all'implementazione elettronica dell'hardware, ma piuttosto quelli volti a conoscere i princìpi teorici di funzionamento e le caratteristiche di calcolo essenziali di questi sistemi di elaborazione. Nello stesso tempo, si è ritenuto utile illustrare brevemente il funzionamento del neurone umano, che costituisce l'esempio di base cui il campo della ricerca informatica si ispira.

Segue a questa introduzione una descrizione un po' più dettagliata delle nozioni basilari connesse con il campo delle reti neurali, tra cui i concetti fondamentali di "neurone elettronico", "peso sinaptico", "topologia neurale" ed "addestramento". Fatto questo, viene introdotto il tipo di elaborazione che una rete neurale artificiale può eseguire, operando, ove possibile, dei parallelismi con analoghe attività cerebrali umane, in modo da chiarire i processi elaborativi con esempi ben comprensibili.

1.2.2 Capitolo 3

Questo capitolo è dedicato allo studio di alcuni modelli di rete neurale artificiale particolarmente semplici e di notevole valore storico. Essi sono utili per comprendere con esempi chiari il funzionamento delle reti neurali artificiali e anche il criterio con cui esse producono le risposte, i loro limiti e le loro caratteristiche, in modo da comprenderne i vantaggi.

Il primo modello presentato, il *percettrone*, è uno dei primi modelli realizzati di questo tipo di sistemi di elaborazione. Di esso si discute in maggior dettaglio la struttura degli elementi costitutivi e il processo di preparazione alla messa in funzione, o "addestramento". Il motivo è che il percettrone costituisce un prototipo tra i più semplici, abbastanza da consentire di seguire i diversi passi di queste fasi essenziali nella predisposizione all'uso di qualsiasi rete neurale artificiale. Vengono poi illustrati i limiti di questo dispositivo, e da qui introdotta una sua evoluzione, che intende mostrare in che modo sia possibile risolvere i problemi connessi con la eccessiva semplicità della struttura proposta all'inizio. Infine, viene proposto un modello implementativo per realizzare una semplice simulazione di questi dispositivi al computer.

1.2.3 Capitolo 4

Questo capitolo si addentra un po' più nello specifico del problema particolare da studiare. Esso introduce, infatti, i concetti qualitativi che è bene conoscere per produrre un algoritmo adatto alla soluzione del problema che la tesi si è prefissato, introducendo una classe di reti neurali, note come memorie auto-associative, e il modello specifico di Hopfield.

Da questo, mediante una serie di passaggi logici, si giunge ai concetti necessari per lo sviluppo dell'algoritmo finale, dedotti attraverso considerazioni che chiamano in causa la meccanica statistica applicata ad un sistema a molti corpi, con l'obiettivo di creare un parallelo tra il suo comportamento ed il funzionamento di una rete neurale di Hopfield in cui si siano introdotte opportune migliorie strutturali. A questo scopo, viene anche presentato un modello fisico esattamente analogo a quello informatico: quello proposto da Ising per studiare in modo semplice i domini ferromagnetici.

Queste considerazioni conducono alla trattazione dell'annealing simulato, che è una tecnica necessaria per rendere la rete di Hopfield capace di fornire una soluzione utile ai problemi che appartengono alla classe generica in cui si inserisce quello della ricostruzione di tracce.

Infine, viene discussa la teoria delle reti MFT (Mean Field Theory), che costituiscono l'evoluzione finale cui si intende giungere e sono la base strutturale e concettuale sulla quale è stato modellato l'algoritmo neurale da impiegare per eseguire il tracciamento delle particelle rivelate nella TPC di ALICE.

1.2.4 Capitolo 5

Questo capitolo introduce e discute l'algoritmo specifico proposto.

Una prima parte illustra in maggior dettaglio il rivelatore che costituisce l'obiettivo dell'applicazione elaborata. Di esso vengono illustrate le caratteristiche tecniche progettuali che sono state prese in considerazione allo scopo di impostare correttamente l'algoritmo proposto.

Nel paragrafo successivo vengono fatte le considerazioni necessarie per adattare il modello generico delle reti MFT al problema in questione, implementandone i concetti in termini di formule matematiche esplicite.

In seguito, viene proposto il programma realizzato, illustrandone il funzionamento, e quindi discutendo le diverse soluzioni sfruttate per ottimizzarne l'efficienza, la semplicità di utilizzo e la rapidità di calcolo.

Dopo, vengono proposti alcuni risultati esemplificativi, basati su simulazioni eseguite con il codice ufficiale in uso all'interno della collaborazione. Su queste simulazioni vengono proposti i risultati della rete, in termini di affidabilità e di efficienza dei risultati, seppure con i limiti di procedure semplificate per il best-fit geometrico delle tracce. Ove possibile vengono eseguiti confronti con altri metodi di ricostruzione delle tracce, per valutare meglio la bontà o meno dei risultati dell'applicazione neurale.

1.2.5 Capitolo 6

Qui vengono tratte le debite conclusioni e le proposte d'applicazione del sistema discusso, in funzione dei risultati, nonché le possibili vie di sviluppo futuro, e tutte le approssimazioni prese in considerazione nelle vari fasi dell'elaborazione.

1.2.6 Appendici

Esistono pure diverse appendici, volte alla discussione dettagliata della matematica che si trova dietro alcune affermazioni proposte nel testo e non sviluppate lungo la sua stesura per non appesantire il discorso. Si dimostrano qui due teoremi utili a giustificare certe caratteristiche delle reti neurali, e le procedure di calcolo necessarie ad estrarre i risultati da ciò che la rete neurale simulata restituisce. Inoltre, essendo stato introdotto un metalinguaggio usato in alcuni paragrafi dedicati alla presentazione di semplici modelli implementativi, un'opportuna appendice ne illustra le specifiche ed i caratteri salienti. Infine, un'ulteriore appendice propone quanto è necessario conoscere per maneggiare il software prodotto in pratica per le analisi di cui la tesi si è occupata.

Capitolo 2

Generalità sulle reti neurali

Volendo indicare, fra i tanti prodotti della tecnologia moderna, quale potrebbe essere il più rappresentativo del progresso avvenuto nel secolo appena trascorso, il pensiero dei più cadrebbe molto probabilmente sul calcolatore elettronico. In svariati campi dell'attività umana, infatti, capita ormai molto spesso che la notevole rapidità di calcolo e l'affidabilità, ai livelli che solo un computer raggiunge, assurgono a "conditio sine qua non" per la realizzabilità di un progetto o per la solubilità di un problema.

Un campo d'applicazione in cui si manifesta con notevole evidenza l'indispensabilità del computer, è proprio quello della Fisica Nucleare e Sub-Nucleare. Si pensi a quanto siano numerosi, in genere, i dati che si registrano in occasione di qualunque esperimento. Si comprende immediatamente quanto poco sia realistico pensare di gestirli senza mezzi informatici, magari operando i calcoli necessari "con carta e penna", per trarne le conclusioni desiderate.

Nonostante le notevoli migliorie che quest'aiutante elettronico consente di apportare al lavoro umano, tuttavia, esso non è esente da limiti. Per esprimerli in una parola, possiamo dire che questa macchina è *veloce*, ma non *intelligente*. Il modo in cui un computer determina la risposta alla domanda postagli, infatti, non consegue ad un "ragionamento", nel senso che comunemente daremmo a questo termine, ma piuttosto ad un "calcolo". Il funzionamento del computer, infatti, consiste essenzialmente nell'esecuzione di un *programma*: una lista d'istruzioni da seguire "alla lettera", una dopo l'altra, fino a determinare una risposta conclusiva. Questo processo, sequenziale e statico, fornisce uguali risposte ad input identici, ma può anche condurre all'impossibilità di ottenerne alcuna, nel caso in cui i dati forniti non siano compatibili con gli standard previsti dal programmatore.

La sfida che i ricercatori informatici hanno raccolto negli ultimi trent'anni è stata, allora, quella di superare questo criterio di elaborazione, per cimentarsi nel tentativo di realizzare un dispositivo elettronico che riesca a "ragionare". L'obiettivo da raggiungere è diventato, cioè, quello di produrre un sistema informatico che fornisca le risposte seguendo un criterio che vada al di là del mero programma, criterio che possa essere definito come una sorta di "logica", pur prendendo le dovute distanze dall'accezione che tale termine assume in rapporto alle facoltà cerebrali umane. L'output ricevuto da strutture organizzate in questo modo, si potrebbe quindi considerare simile ad una "deduzione". Naturalmente, questo tipo di ricerca non intende prefiggersi di produrre un vero e proprio "cervello elettronico", anche perché il funzionamento dell'encefalo umano è un campo di conoscenza scientifica che si è ancora ben lungi dall'aver esaurito di studiare.

Uno dei risultati attuali cui si è giunti è la produzione di sistemi informatici costituiti da numerosi processori elementari collegati fra loro, che si scambiano le informazioni secondo

un criterio molto vicino a quello con cui lo stesso fenomeno avviene tra i neuroni fisiologici. Proprio questa somiglianza giustifica la denominazione di **reti neurali artificiali** (RNA) con cui si suole indicarli. La strutturazione e, come si vedrà più avanti, il principio di funzionamento delle reti neurali artificiali, comportano diversi vantaggi [Bol95] che mostrano la loro utilità in vari campi d'applicazione, i quali, generalmente, mettono a dura prova anche il più potente degli elaboratori "tradizionali". Ciò succede, ad esempio, quando la quantità delle variabili da cui dipende un problema è troppo grande, oppure quando i tempi di analisi sono molto lunghi, o ancora quando è difficile, a causa della complessità del caso in esame, dedurre le corrette funzioni deterministiche da applicare, e dalle quali trarre gli algoritmi efficaci da implementare al computer.

Un grosso vantaggio quantitativo delle reti neurali artificiali, il cui apporto influenza principalmente la rapidità del calcolo, è il **parallelismo** di quest'ultimo. Il fatto che la RNA sia costituita da diversi piccoli elaboratori collegati, anziché uno solo, comporta una "divisione dei compiti" tra essi, determinando uno sviluppo in parallelo del calcolo, che è evidentemente più rapido di quello sequenziale del computer. In quest'ultimo, infatti, l'unica CPU presente deve compiere tutte le operazioni richieste dal programma, ma, potendone eseguire una alla volta, deve funzionare per una quantità complessiva di tempo che è direttamente proporzionale alla quantità dei calcoli previsti.

Dal punto di vista del risultato, la caratteristica basilare di una rete neurale artificiale si sintetizza nella parola **generalizzazione**. Una RNA, infatti, analizza i dati in modo tale da permetterle di restituire una risposta in ogni caso. Anche quando le si presentasse un input imprevisto, una rete non potrebbe bloccarsi o generare errori: il massimo rischio sarebbe quello di trovare una risposta "strana", avendo analizzato un problema secondo una "logica" non adatta ad esso. Il motivo è che, come si vedrà, il funzionamento della rete è simile al processo di riconoscimento che una qualsiasi sensazione esterna innesca nel cervello umano, e, come questo, conduce sempre ad una possibile ricostruzione, anche quando può rivelarsi errata (si pensi a quando, osservando un oggetto in lontananza, si commettono errori sulla valutazione della sua natura o delle sue dimensioni).

Un ultimo aspetto caratteristico delle reti neurali artificiali, riguardante stavolta il modo di immagazzinare i dati necessari al suo corretto funzionamento, è ciò che si chiama **rappresentazione distribuita**. Questo è il modo in cui si indica la proprietà, tipica di simili sistemi, per cui tutto ciò che ad essi serve per operare correttamente è contenuto in tutto l'insieme, ossia non esiste un processore principale, che, da solo, ne condiziona il corretto funzionamento. Piuttosto, la "logica" in base alla quale la RNA esegue le sue analisi è insita nel complesso della sua stessa struttura, e quindi essa opererebbe in modo soddisfacente anche quando uno dei suoi elementi costitutivi dovesse danneggiarsi. In realtà ciò succede anche all'encefalo umano, in cui alcuni neuroni, col tempo, perdono parte delle propria funzionalità, ma non possono essere sostituiti, dato che ciò comporterebbe la necessità di acquisire nuovamente tutte le conoscenze possedute (comprese le associazioni più banali, come i colori con i loro nomi, e le facoltà di base, come il parlare o lo scrivere). La rete neurale artificiale, come è chiaro da questo concetto, conserva nella sua stessa struttura la "logica" che ne governa il funzionamento. Ciò contrasta abbastanza sonoramente con le condizioni di un computer, il cui funzionamento è tutto concentrato in un programma scritto e memorizzato opportunamente: è chiaro che un danneggiamento anche minimo di esso, lo rende inservibile, e, se è un programma di base per il funzionamento della macchina intera, rende inutile anch'essa fino ad un suo completo ripristino.

L'applicazione di cui si intende discutere vuole sfruttare queste considerazioni per proporre un criterio di analisi al computer che, seppur a livello di simulazione, si avvalga

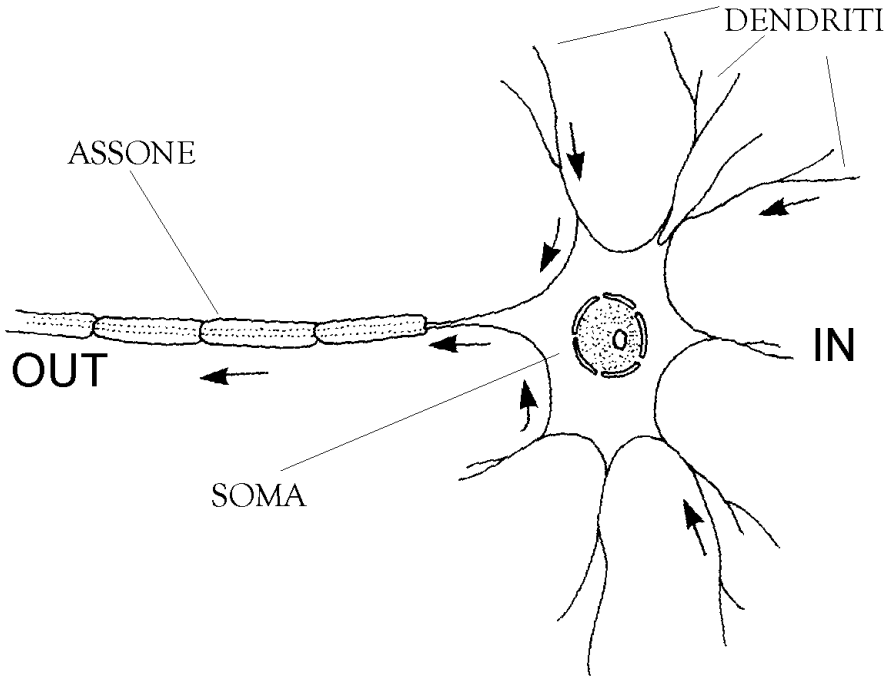

Figura 2.1: Un neurone umano. Le frecce indicano il senso in cui viaggiano i segnali elettrici per lo scambio di informazioni.

dei caratteri fondamentali di "data processing" delle reti neurali. Prima di procedere, però, è opportuno ed interessante, allo scopo di comprendere appieno il funzionamento di questi sistemi informatici, soffermarsi a richiamare alcuni semplici concetti di fisiologia ed anatomia dei neuroni umani, che ne sono il modello naturale.

2.1 Il neurone umano

Il sistema nervoso umano è il mezzo attraverso cui il cervello gestisce e controlla tutte le funzioni vitali, volontarie e non, dell'organismo. Esso consente la trasmissione di informazioni provenienti dall'esterno, mediante i ricettori sensoriali, come anche l'invio delle disposizioni necessarie a modificare il funzionamento di ogni parte del corpo, per adattare l'organismo ad una specifica situazione. Inoltre, il cervello, cui l'intero sistema fa capo, è il coordinatore di tutte le operazioni di immagazzinamento e classificazione delle informazioni che il corpo riceve dall'esterno. La cellula che costituisce il "mattone" di questa potente struttura anatomica, e che funge da modello per gli apparati informatici di cui si intende trattare in questa tesi, è il **neurone**.

2.1.1 Struttura anatomica e fisiologia. Analogie elettroniche

La figura 2.1 mostra un'immagine schematica del neurone umano.

Il neurone è un mezzo di trasmissione di informazioni capace di far viaggiare un impulso elettrico senza sostanziale perdita d'intensità anche per quasi metà dell'altezza del corpo umano (si pensi che il corpo cellulare principale che presiede al moto dei piedi si trova più

in alto dell'osso sacro). Come mostra la figura 2.1, un neurone consta di tre sezioni distinte [Bol95]:

Dendriti. Sono ramificazioni brevi e numerose, che ricevono dalle altre unità o dai ricettori sensoriali esterni i segnali che costituiscono l'input;

Soma. È la regione in cui viene elaborato l'insieme di tutti gli input ed è prodotto un output: esso è, in pratica, la CPU del neurone;

Assone. Consiste in un unico canale che inoltra l'output verso altri neuroni.

Il punto di collegamento tra l'assone di un neurone e un dendrite di un altro — ma è possibile anche un collegamento diretto assone-soma — si chiama **sinapsi**, e può consistere in un contatto diretto tra le due cellule, o anche in una regione di estrema vicinanza tra le loro membrane.

Da analisi compiute sperimentalmente [Rin97], si mostra che il funzionamento delle cellule neurali umane è basato sullo scambio di segnali elettrici estremamente deboli. Essi sono generati dalla differenza di potenziale che si instaura tra l'interno della cellula e la regione esterna nelle sue immediate vicinanze, a causa di squilibri nelle concentrazioni di diversi tipi di ioni positivi, presenti in questi due spazi.

Quando il neurone è inattivo, l'interno della cellula è a potenziale minore dell'esterno, di circa 70 mV (*potenziale di riposo*), come se ci fossero, rispettivamente all'interno ed all'esterno, allineamenti di cariche negative e positive, tanto da giustificare l'espressione "membrana polarizzata". Quando insorge l'eccitamento neurale, la polarizzazione si inverte rapidamente, portando l'interno della cellula ad un potenziale di circa 30 mV superiore rispetto all'esterno (*potenziale d'azione*), dopo di che, in mancanza di ulteriori stimoli, il sistema ritorna in quiete. Ciò che avviene in pratica, quindi, è la produzione di un impulso di potenziale molto simile, anche se molto più debole, a quello prodotto da un normale rivelatore di particelle usato in Fisica Nucleare. Sostanzialmente, questi cambiamenti di differenza di potenziale (d.d.p.) tra l'interno e l'esterno del corpo cellulare sono dovuti ad uno scambio di ioni, ad alcuni dei quali la membrana cellulare del neurone è permeabile. Gli ioni interessati dal processo sono tutti positivi, ma la loro differente natura — ed elettronegatività — produce queste microscopiche variazioni di potenziale, che possono essere eventualmente accentuate da un opportuno aggiustamento delle concentrazioni. Il fenomeno di eccitazione viene consentito da un noto fenomeno fisico: lo spostamento degli ioni, in presenza di membrane semipermeabili, in seguito alla presenza di un gradiente di concentrazione non uniforme. È noto come, in questa circostanza, gli ioni interessati tendano a spostarsi secondo tale gradiente, anche a scapito dell'omogeneità della distribuzione delle cariche, il che causa l'insorgere di d.d.p.. Per avere un'immagine chiara del processo, si potrebbe proporre l'analogo fisico della corrente di spostamento in una giunzione di semiconduttori *n-p*, che sta alla base del funzionamento di qualunque rivelatore a stato solido.

Naturalmente, se l'eccitazione del neurone avviene con una variazione spontanea delle concentrazioni, il processo inverso, che si manifesta in occasione del "rilassamento", è invece imposto "a forza" da opportuni, ma non ancora troppo noti, elementi cellulari, chiamati *pompe*, i quali costringono gli stessi ioni a muoversi in senso opposto a quello che sarebbe loro naturale, mentre si riduce la permeabilità della membrana cellulare che si era in precedenza lasciata attraversare da essi. Da quanto detto, risulta abbastanza evidente come il reale segnale elettrico innescato e trasmesso da un neurone sia dovuto essenzialmente

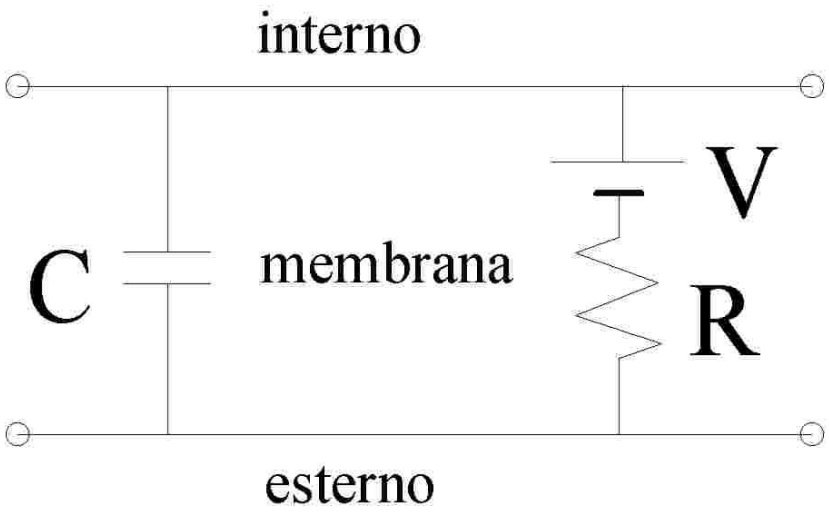

Figura 2.2: Equivalente elettrico di un neurone. La membrana rappresenta un mezzo capacitivo, ai capi del quale è presente una d.d.p.. La resistenza è rappresentata dal fatto che la permeabilità parziale e le pompe controllano il flusso degli ioni e si oppongono in parte ad esso.

all'innesco ed alla susseguente diffusione di tale differenza di potenziale tra l'interno e l'esterno della cellula stessa. Possiamo immaginare, semplificando molto il problema, un semplice circuito RC con generatore, come in figura 2.2, per voler rappresentare un modello elettronico di questo sistema.

Quest'immagine schematica fornisce già alcuni parametri in base ai quali progettare una struttura che, in forma semplificata, possa emulare il funzionamento elettrico del neurone. Quanto alla forma della risposta, ciò che si è sperimentato è che essa è estremamente rapida ("spike signal"), ma uno stimolo eccitativo, per produrne una, deve sempre essere superiore ad una certa soglia, al di sotto della quale il neurone non reagisce. Per di più, la risposta di un neurone a qualsiasi stimolo superiore alla soglia è la stessa, nel senso che l'altezza del segnale non dipende da esso (legge del "tutto o niente"). Altra informazione importante che ha un analogo nei circuiti elettronici in uso comune in Fisica, è la presenza di un "tempo morto", durante il quale il neurone non è sensibile agli stimoli. Il motivo di questo tempo morto è la necessità di riassestare le concentrazioni ioniche, di cui si è detto sopra, nello stato di quiete, per essere pronto a ricevere una nuova eccitazione esterna.

Da quanto visto finora, mantenendo comunque la discussione entro termini molto semplificati, il funzionamento di risposta del neurone è molto simile a quello di un rivelatore: arrivo di un segnale, risposta in forma di impulso di tensione elettrica, tempo morto necessario al riassestamento e allo smorzamento del segnale prodotto.

Un altro punto importante del funzionamento del neurone, che è d'interesse fondamentale nella discussione del modello fisiologico per le reti neurali artificiali, è la propagazione del segnale. Infatti, il funzionamento del sistema nervoso prevede che un neurone eccitato comunichi questo suo stato d'attivazione ad altre unità, o per portare un'informazione al cervello, o nel senso inverso, quando è il cervello ad "impartire un ordine" ad una parte del corpo. La comunicazione avviene attraverso l'assone, come si è visto, ed è dovuta essenzialmente alla tendenza, tipica del potenziale d'azione (la d.d.p. in stato di eccitazione), a propagarsi centrifugamente, inducendo una depolarizzazione nelle aree della membrana

neurale adiacenti a quella in cui è stato innescato. In questo modo, avviene un processo di propagazione della d.d.p. indotta inizialmente, in un modo che ancora una volta ha un analogo, anche se estremamente semplificato, nell'elettronica che la Fisica adopera: il cavo coassiale.

Il segnale può essere trasmesso in due modi:

- una semplice ed immediata trasmissione della d.d.p. da un neurone all'altro, quando la connessione sinaptica è di contatto diretto;
- un processo combinato in cui alcune sostanze chimiche atte a trasmettere l'informazione (ad esempio, l'ordine del cervello di contrarre un muscolo), viaggiano entro l'assone, e vengono liberate verso il soma del nervo cui il segnale è diretto, solo dopo che alla sinapsi è giunto un segnale in tensione che funge da "via libera"; in questo caso non c'è contatto diretto tra i due corpi cellulari.

La comunicazione può avere il requisito di essere più rapida o più precisa, caratteristiche che non possono essere conseguite entrambe allo stesso livello. Nel caso di connessioni rapide, esistono delle guaine di una sostanza chiamata *mielina*, le quali ricoprono alcuni tratti dell'assone, impedendo in queste zone la diffusione ionica che trasmette il segnale. L'effetto è che il segnale si propaga "a salti", tra le regioni non ricoperte dalla mielina, in modo da essere rapido ma non troppo preciso. Invece, nel caso in cui la necessità di precisione debba prevalere sulla rapidità, le guaine di mielina sono assenti, per cui la propagazione è continua, ma più lenta.

L'azione del segnale trasmesso può essere duplice: stimolare l'attivazione del neurone che funge da bersaglio, oppure inibirla, nel qual caso, al neurone che riceve il segnale, viene impedito di attivarsi anche in presenza di stimoli. Questo concetto è importante nello sviluppo delle reti neurali artificiali. Può capitare, infatti, che esistano alcuni vincoli che condizionano il problema da affrontare, imponendo quindi di scartare certe configurazioni delle attivazioni neurali. Tutto ciò sarà più chiaro quando saranno stati introdotti i concetti generali di cui si parlerà nei paragrafi successivi. Il motivo per cui è stato proposto qui questo discorso è quello di introdurre un elemento molto importante circa la comunicazione. Essa non consiste semplicemente nella trasmissione di un segnale, ma, più in generale, nella ricombinazione di tutti i segnali ricevuti entro un determinato tempo, la quale determina l'output inoltrato in avanti. Il fatto che ogni neurone rielabori il suo input, assieme al fatto che la trasmissione può avere carattere tanto inibitorio quanto eccitativo ed un valore diverso per ogni cellula, si traduce, in un'immagine elettronica quale è quella che si sta proponendo, con la presenza di un *peso* per ogni connessione. In altre parole, si può definire un valore numerico, associato ad ogni connessione neurale, che moltiplica l'intensità di ogni segnale che l'attraversa. L'effetto di questo valore, chiaramente, secondo il segno e il modulo, può essere quello di invertire, amplificare o smorzare il segnale in partenza. In particolare, una connessione associata ad un peso negativo è evidentemente inibitoria, dato che un neurone "acceso" invia un segnale polarizzato in senso opposto allo stato di accensione, e quindi tende a "spegnere" il ricevente.

È il caso di chiudere questo paragrafo con un esempio pratico della fisiologia delle connessioni inibitorie ed eccitative. Ogni punto del corpo che sia sensibile alle percezioni tattili possiede, in realtà, due canali di trasmissione: una sensoriale ed una del dolore, ed entrambe inviano il loro stimolo ad un neurone che si definirà "di smistamento". Quest'unità, oltre ad inviare il segnale al cervello, ha due connessioni inibitorie di feed-back con il ricettore del tatto e quello del dolore. Quello che succede, dunque, è che quando giunge

sul ricettore uno stimolo combinato di percezione tattile e dolore, prevale quello che ha una durata maggiore, in quanto il segnale prodotto da ciascun ricettore ed inoltrato al neurone di smistamento, provoca l'inibizione dell'altro segnale, finché uno dei due elementi in competizione non rimane in quiete per l'assenza di stimoli successivi. Questo è il motivo per cui frizionare una parte dolente aiuta a bloccare la sensazione del dolore.

2.1.2 Leggi della conduzione dell'impulso

Oltre a quella del "tutto o niente" esistono altre regole empiriche che il sistema nervoso sembra rispettare nella strutturazione delle comunicazioni [Rin97]:

Legge dell'integrità anatomica. Un neurone può essere utilizzato solo se è perfettamente integro, perché, quando è rotto o fortemente deformato, non funziona nemmeno se se ne giustappongono le parti.

Legge della conduzione isolata. Ogni conduzione nervosa macroscopica contiene molti assoni paralleli, ma ciascuno non influenza gli altri con il proprio segnale, per cui ogni comunicazione neurone-neurone è isolata.

Legge della conduzione in avanti. Stabilisce come un segnale venga trasmesso solo in un senso, per cui un assone è una comunicazione elettrica unidirezionale, la qual cosa assicura il corretto funzionamento del sistema.

Legge della conduzione bidirezionale. Sebbene in contrasto con la precedente, viene talvolta applicata in elettronica, e vale per i sistemi fisiologici solo in potenza, nel senso che l'assone *è comunque in grado* di comunicare in entrambe le direzioni.

Legge degli effetti costanti. Questa legge, oltre a prevedere che la risposta di un neurone è sempre la stessa, assicura che viene inviata sempre ai medesimi elementi; si potrebbe ridefinire, quindi, questa legge come una "legge dell'invariabilità della topologia neurale", la quale è di fondamentale importanza nell'ottica della realizzazione di un modello artificiale di questo complesso sistema fisiologico.

A queste regole è utile aggiungere i risultati di uno studio pionieristico di McCulloch e Pitts [Free91], volto proprio a comprendere come questi semplici elementi neurali, anche se passibili di innumerevoli variazioni, possano condurre a tutte le capacità del cervello. É notevole il fatto che il loro lavoro sia il luogo in cui, per la prima volta, il cervello viene trattato alla stregua di "calcolatore organico". Lo studio di McCulloch e Pitts si fonda sulle stesse regole elencate in precedenza e su alcune ipotesi aggiuntive:

1. un neurone si accende solo se viene messo in funzione un certo numero di sinapsi entro un determinato periodo di tempo, detto di *addizione latente*, che è quello disponibile prima di innescare la risposta dello stesso;

2. l'unico ritardo significativo nel transito delle informazioni è quello dovuto allo scambio di informazioni tramite le sinapsi;

3. quando si attiva una sinapsi inibitoria, ciò determina senz'altro lo spegnimento dell'unità;

La regola del "tutto o niente" lascerebbe pensare che l'attivazione di un neurone debba essere rappresentata, seguendo una terminologia informatica, da una *variabile booleana*, (quelle del tipo "vero/falso"), il cui valore dipenda da un operatore, booleano anch'esso, delle variabili corrispondenti alle attivazioni dei neuroni di input. Tuttavia, in realtà, esistono modelli elettronici in cui l'attivazione è un numero reale, e saranno utili per alcuni modelli precisi di sistema neurale artificiale. Mantenendo momentaneamente il discorso nel caso di attivazioni binarie, comunque, si può giustificare l'estrema versatilità del sistema cerebro-neurale umano pensando che è possibile costruire 4^n differenti funzioni binarie di n variabili binarie, e che nel caso in questione i valori di n sono estremamente grandi (solo nel cervello ci sono circa 10 miliardi di unità). Considerare poi attivazioni reali, non serve ad altro che ad aumentare ulteriormente questo numero.

È chiaro che tutto il discorso finora presentato non esaurisce la complessità del funzionamento di queste complicatissime strutture. Esso è semplicemente un'illustrazione delle proprietà che è utile conoscere per possedere un modello a cui ispirarsi, nell'intento di riprodurre un apparato elettronico che emuli il procedimento di calcolo di una rete di neuroni.

2.1.3 Addestramento

Come è noto, il cervello non nasce già fornito delle proprie conoscenze. Esso le acquisisce col tempo, classificando ed immagazzinando nella memoria le percezioni che riceve dal momento della nascita in poi. Si pone il problema di capire come faccia, in termini della struttura del circuito neurale, a immagazzinare queste informazioni.

Fu Hebb [Free91] a trovare la risposta alla domanda "Come impariamo?", partendo da un'assunzione ancor oggi sfruttata nell'addestramento delle reti neurali. Considerando che il sistema neurale sia posto in un certo stato di attivazione complessivo a causa di una percezione, la regola di Hebb prevede che l'intensità della connessione sinaptica fra due neuroni che si trovano nello stesso stato d'attivazione venga incrementata, e si riduca, invece, quella tra due di essi che tendano a trovarsi in stati di attivazione opposti.

Un esempio storico della validità di questo asserto è il cane di Pavlov, che, dopo essere stato abituato a mangiare al suono di una campanella, aveva associato i due impulsi (suono e cibo) al punto da salivare abbondantemente al semplice suono della campanella, anche senza il cibo, in quanto la stimolazione dei ricettori sensoriali uditivi e quella della salivazione erano state messe strettamente in relazione dalle numerose prove fatte sull'animale.

Questo modello, come del resto quello precedente di McCulloch e Pitts, non sono estremamente precisi, ma hanno un'importanza consistente, in quanto ad essi si ispirano criteri di realizzazione ed addestramento di alcuni modelli pionieristici di reti neurali artificiali.

Dopo aver illustrato questi concetti generali sulla struttura del neurone e sui criteri di connessione, comunicazione ed addestramento, è possibile introdursi con un po' più di cognizione di causa nel campo specifico delle RNA. Il primo passo è costituito evidentemente da alcune considerazioni generali sugli elementi costitutivi, fisici e concettuali, di questi sistemi.

2.2 Concetti generali sulle RNA

Le reti neurali non hanno la pretesa di riprodurre esattamente un sistema nervoso, anche perché le conoscenze in questo senso non sono sufficientemente evolute e la complessità di un cervello umano è estremamente maggiore di quella del calcolatore più potente che si possa pensare, al giorno d'oggi, di realizzare. Nell'introduzione, si è detto come una rete neurale sia un sistema che applica ad un problema un criterio di analisi basato non su di un unico e potente processore, ma bensì su una struttura costituita da unità semplici ed interconnesse. Conoscendo la struttura del sistema nervoso, è adesso più chiaro il motivo di questa scelta, almeno nella misura in cui si voglia imitare, molto in piccolo, il criterio di "data processing" del modello fisiologico da cui quello informatico in esame prende il nome. Il fatto che non si riesca a creare un vero e proprio analogo elettronico del cervello è evidente dalla molteplicità dei modelli di rete neurale esistenti, ciascuno dei quali si avvicina ad *alcuni* aspetti del modello umano, senza avere mai la pretesa di riprodurli tutti esattamente.

Gli elementi essenziali che caratterizzano un certo tipo di rete neurale artificiale sono essenzialmente tre [Hum90]: il tipo di neurone utilizzato, le modalità di connessione interneurale e la forma di "comportamento intelligente" compiuta sui dati in ingresso, per quanto quest'ultimo parametro non è da intendere esattamente secondo la sua accezione più comune. Questi tre aspetti strutturali verranno descritti più in dettaglio nei prossimi paragrafi.

2.2.1 Elementi fondamentali

In base alle caratteristiche essenziali viste nel neurone umano, si comprende, in linea di principio, che operazioni debba essere in grado di eseguire un suo analogo elettronico:

1. ricezione e somma di diversi segnali elettrici esterni, da connessioni di intensità fissata in modo da riprodurre i pesi sinaptici fisiologici;

2. operare una trasformazione di quest'input complessivo;

3. produrre un'unica risposta da inoltrare verso altre unità del sistema.

Il punto 1 pone l'accento sul fatto che, se il neurone elettronico determina sostanzialmente la tipologia della RNA ed il suo comportamento, una discussione intesa ad illustrarne il funzionamento non sarebbe completa se non si menzionassero le *connessioni sinaptiche*, ossia i mezzi elettronici attraverso cui i neuroni comunicano. Ciascuna di queste connessioni è unidirezionale, il che implica un flusso delle informazioni a senso unico, a meno che non si provveda esplicitamente a connettere due unità in entrambi i sensi. Il ruolo essenziale delle connessioni non è solo quello di trasferire i segnali, ma anche quello di regolarne l'intensità e la polarità, per cui è importante associare a ciascuna un numero reale che ne rappresenta il peso. Come si è visto nel caso dei neuroni umani, infatti, è proprio in questo valore che si concretizza la "conoscenza" della RNA, per cui esso è ciò che, in ultima analisi, stabilisce le risposte che la rete produce dai dati ricevuti. Naturalmente, una connessione eccitativa sarà rappresentata da un valore positivo, che, seppure possa amplificare (se è > 1) o smorzare (se è < 1) un segnale in ingresso, veicola comunque l'unità ricevente ad uno stato di attivazione concorde con quello della sorgente del segnale. Al contrario, una connessione inibitoria sarà rappresentata da valori negativi, che tentano di inviare all'unità ricevente

un segnale di polarità inversa alla propria: un'unità attiva manda segnali che tendono a spegnere il neurone che li riceve, e viceversa.

Ogni unità riceve diversi segnali in ingresso e li somma per ottenere il valore da inviare alla regione preposta all'elaborazione. Quest'ultima produce un risultato che costituisce l'output dell'unità presa in considerazione, il quale viene inoltrato verso tutti i neuroni che attendono input da essa. Per tenere conto del peso sinaptico di ogni connessione, la somma dei segnali che produce l'input complessivo deve essere pesata su questi valori. Ipotizzando di ordinare gli elementi della rete, in modo da associare un indice intero ad ogni neurone, si può rappresentare il peso della connessione tra l'unità i-esima e la j-esima con la scrittura w_{ij}. Se, per esempio, si indica con l'indice i il generico elemento che invia il suo output x_i alla j-esima unità, l'input complessivo ricevuto dal neurone j risulta essere il seguente:

$$net_j = \sum_i w_{ij} x_j \tag{2.1}$$

Questo è il valore che la j-esima unità elaborerà producendo l'output da inoltrare verso altri neuroni.

Nello studio del processo secondo cui si sviluppa l'iter che conduce dalla ricezione alla produzione dell'output, il primo passo consiste nel determinare l'attivazione di un neurone in conseguenza dell'input ricevuto. Questo è il compito della funzione di attivazione, indicata generalmente con F. Essa ha certamente come argomento, per l'appunto, l'input complessivo, ma può tenere conto, in certi casi, anche dell'attivazione precedente. Il valore restituito da F è quello che si suole indicare come l'attivazione dell'unità in questione. Una volta che si sia stabilita l'attivazione di un neurone, da questa esso deve determinare il segnale che sarà inviato a quelli che lo seguono nella catena sinaptica della rete neurale. Questo è il compito della funzione di output, indicata con f. Seguendo la notazione usata nella (2.1), quindi, $x_i = f(a_i)$. In genere, le forme analitiche che f può assumere [Hum90] sono due:

1. l'identità, o una funzione lineare, nel caso in cui il neurone trasmetta semplicemente la propria attivazione, a meno di un fattore di proporzionalità;

2. una funzione a soglia, in cui il neurone invia un segnale solo se la sua attivazione supera un valore minimo assegnato.

È chiaro, parlando di F ed f, che il presupposto è quello di usare, salvo casi particolarissimi, neuroni dello stesso tipo, caratterizzati da uguali funzioni di attivazione e di output.

A questo punto, è chiaro il modo in cui una rete elabora le informazioni. Innanzitutto si dovranno individuare, con criteri diversi in base a caratteristiche topologiche che si vedranno più avanti, una regione di input che è preposta a ricevere dall'esterno il segnale da elaborare ed una regione di output, da cui viene estratta la risposta della rete. Inizialmente, quindi, vengono fissate le attivazioni degli elementi della regione di input, determinate a partire dal significato fisico che essi hanno. Fatto questo, si lascia evolvere la rete, facendo fluire le informazioni, a causa delle quali i diversi neuroni modificano, una o più volte, secondo il tipo di rete, la propria attivazione. A processo ultimato, viene raccolta la risposta traducendo lo stato d'attivazione dei neuroni della regione di output nel suo significato fisico. Osservando il processo, tra l'altro, si comprende perché non esistano, qui, problemi come incompatibilità od errori di elaborazione, che talvolta si presentano quando ad un comune calcolatore sono forniti dati non conformi ai canoni previsti da chi ha progettato la macchina o realizzato il programma in esecuzione. Al contrario, nel caso

delle RNA, una volta che si traduce l'input in uno schema di attivazioni iniziali, la risposta è garantita in ogni caso.

Viste le caratteristiche salienti di un neurone, si comprende come esso sia completamente descritto dalle sue funzioni caratteristiche f ed F. La prima, si è detto, assume in genere una forma semplice (identità o risposta a soglia). La seconda, invece, può assumere forme molto diverse, per rispondere opportunamente ai diversi problemi analitici che una rete neurale artificiale potrebbe essere chiamata ad affrontare. È utile allora osservare alcuni tra gli esempi più diffusi di funzione d'attivazione, premettendo che esse si distinguono in due grandi classi: quelle discrete e quelle continue. Le prime sono funzioni che possiedono un valore di soglia, con cui confrontano l'input totale ricevuto, restituendo un'attivazione uguale a 1 se il primo è maggiore della seconda, e a 0 o -1 in caso contrario.

Le funzioni continue, a differenza delle precedenti, pongono l'input ad argomento di una funzione analitica, il cui valore è l'attivazione dell'unità. Tra queste ultime, poi, ne esistono alcune limitate, che mantengono i valori entro un preciso intervallo, ed altre che non hanno limiti per il valore delle attivazioni.

Tra le funzioni d'attivazione non limitate, il caso più interessante è quello in cui F è lineare. In questo caso, l'attivazione del neurone è semplicemente il suo input complessivo. In aggiunta, può essere considerato un termine di soglia S, tale da azzerare l'attivazione se l'input ricevuto non supera S.

Tra le funzioni d'attivazione continue e limitate, troviamo elementi rappresentativi nel neurone termodinamico, in quello di Grossberg [Hum90] ed in quello ad attivazione interattiva [Hum90]. Il neurone termodinamico è un dispositivo ad attivazione binaria (0 o 1), in cui tale attivazione viene definita, però, con un processo casuale, governato da una distribuzione statistica non uniforme, di tipo sigmoidale. Questo tipo di neurone sarà meglio illustrato in seguito, in quanto il suo funzionamento è una tappa essenziale verso la produzione dell'algoritmo neurale la cui applicazione si intende proporre in questo lavoro di tesi.

Il neurone di Grossberg e quello ad attivazione interattiva sono caratterizzati da una funzione particolarmente complessa, che prevede di considerare diversamente un input eccitativo da uno inibitorio, nonché di smorzare progressivamente l'attivazione, a partire dal suo ultimo valore. È opportuno scrivere la forma della funzione di attivazione F, vista la forma complessa che presenta in questo caso:

$$F = a(1-d) + (B-a)I_{ecc} + (a-A)I_{inib} + a_0 d \tag{2.2}$$

Nella formula (2.2, a è l'attivazione che il neurone aveva precedentemente, A e B sono gli estremi inferiore e superiore dell'intervallo dei possibili output, I_{ecc} ed I_{inib} rappresentano i contributi eccitativo ed inibitorio totali, d è il termine che controlla lo smorzamento di cui si è detto ed a_0 è un valore che funge da "livello zero". La differenza tra i due tipi di neurone sta nel modo in cui vengono calcolati I_{ecc} ed I_{inib}. Nel neurone di Grossberg, vengono ottenuti separatamente dagli input provenienti da connessioni con peso positivo e negativo rispettivamente, quindi compaiono entrambi in ogni caso, mentre nel neurone ad attivazione interattiva la somma viene fatta su tutti gli input, per cui, tra I_{ecc} ed I_{inib} ne compare sempre uno solo. Questi modelli servono anche a comprendere in che modo può succedere che la nuova attivazione di un neurone tenga conto della vecchia, per cui l'input ricevuto non serve tanto a "modificare", quanto piuttosto a "spostare" il valore dell'attivazione.

Per completare il quadro, è bene menzionare un tipo particolare di neurone, detto *connectionist unit*, che ha una funzione di output particolare, non proprio a gradino, ma

comunque con la possibilità di restituire solo valori discreti compresi in un intervallo. In pratica esso esegue un processo di arrotondamento dell'attivazione reale. Le funzioni d'attivazione che si possono usare sono molteplici, in quanto è applicabile qualunque di quelle precedenti viste, ma una forma molto comune è quella della P-unit, in cui la nuova attivazione è uguale alla vecchia più un multiplo dell'input complessivo, finché essa non raggiunge un valore massimo, sul quale, raggiuntolo, si stabilizza.

2.2.2 Schemi topologici

È importante definire il modo in cui si connettono tra loro le unità in una rete neurale, in quanto ciò influenza il flusso delle informazioni e l'azione che ogni unità esercita sulle altre. Inoltre, la topologia della rete neurale è ciò che individua la collocazione delle regioni di input e di output, che possono coincidere o meno.

Una maniera di strutturare la rete è quella della disposizione a strati. Questa struttura topologica prevede il raggruppamento di tutti i neuroni in insiemi, detti strati perché graficamente, in genere, vengono rappresentati come righe successive in una struttura a più livelli. Quanto alle interconnessioni, esse vengono stabilite solo tra i neuroni di due strati consecutivi, in modo che il flusso delle informazioni passi per uno strato alla volta, ed in un determinato senso, data l'unidirezionalità delle connessioni stesse (feed-forward networks). Naturalmente, in una rete strutturata con questo criterio, il primo strato della sequenza sarà quello di input e l'ultimo quello di output, e il lavoro della rete consisterà essenzialmente nel trasmettere, a partire dallo strato di input, le informazioni, tradotte in attivazioni neurali, estraendo all'altro capo della catena la risposta. Ciò comporta un'elaborazione abbastanza rapida che prevede un solo aggiornamento per ogni neurone, corrispondente al passaggio dell'informazione attraverso lo strato cui esso appartiene, ciascuno dei quali la elabora una volta sola. Oltre allo strato di input e di output, naturalmente, possono essere presenti altri strati, detti "nascosti" in quanto, in effetti, il loro stato di attivazione non viene visto dall'esterno, ma serve solo da parametro transitorio per ottenere una risposta più affidabile e capace di distinzioni più fini tra esempi simili. È da notare che unità nello stesso strato non comunicano fra loro, ma interpretano solo lo stato complessivo dello strato precedente, sulla base dei pesi sinaptici delle connessioni realizzate con esso.

Un altro tipo di topologia è la connessione completa. In questo modello, i neuroni sono disposti in un unico strato, e ciascuno è collegato con tutti gli altri ed ha, a volte, anche una connessione di feed-back con se stesso. Naturalmente, quest'unico strato funge contemporaneamente da input layer e da output layer. Il modo in cui viene elaborata l'informazione da questa topologia, ovviamente, è un processo di auto-aggiornamento, in cui le attivazioni di tutte le unità si modificano ripetutamente, finché non si giunga ad una situazione in cui nessuna varia più. A questo punto, l'insieme delle attivazioni della rete rappresenta la sua risposta.

Le due diverse topologie, più che competere in efficienza e adattabilità, sono complementari. In altre parole, in base al tipo di elaborazione richiesta, sarà più utile la prima o la seconda, senza che si possa parlare espressamente di confronto fra le due. Esiste infine una topologia particolare [Free91], che combina le due. In questa disposizione, esistono due strati, detti strato visibile e strato nascosto. Ogni neurone dello strato visibile è collegato con tutti quelli dell'altro, con connessioni bidirezionali, che servono solo a trasferire l'informazione a quest'ultimo e a trarne in seguito la risposta. Lo strato nascosto, poi, è completamente interconnesso, come le reti ad un solo strato, e come queste elabora le informazioni ricevute, fino a produrre una configurazione stabile. Fatto questo, le connes-

sioni tra i due strati vengono riutilizzate al rovescio e forniscono un set di attivazioni dello strato visibile che costituisce la risposta finale della rete.

2.2.3 "Intelligenza" della rete

Avendo premesso che una rete neurale artificiale non intende riprodurre fedelmente un cervello umano, ma solo alcune delle sue facoltà, si comprende come bisogni adattarla opportunamente a quella che interessa sfruttare. Per questo esistono diversi modelli, tra i quali va selezionato, volta per volta, il più adatto a riprodurre un preciso tipo di analisi. Nonostante la loro grande varietà, tuttavia, è possibile classificarli in insiemi, sottolineando i diversi tipi di "comportamento intelligente" che una rete può manifestare.

Sostanzialmente, si può esprimere l'operazione di qualunque rete neurale con una parola: *classificazione*. Infatti, qualunque rete basa il proprio funzionamento sul concetto di ricevere un input, e produrre un output che costituisce una "interpretazione" di esso. L'input rappresenta sempre una combinazione di segnali che, seppure non debbano essere necessariamente ottici, li si denominerà "immagine", intendendo con essa l'insieme dei segnali provenienti da un oggetto fisico, che contengono informazioni su alcune sue caratteristiche e ne consentono il riconoscimento. Una RNA è, a livello concettuale, un dispositivo che "conosce" alcune di queste immagini, e così, quando riceve un esempio mai visto, cerca di ricondurlo ad uno di quelli ad essa noti. Dal momento che le "conoscenze" di una rete neurale artificiale dipendono dai parametri che essa in pratica analizza in un oggetto, ne consegue che esse rappresentano intere classi di elementi, per cui il processo, in ultima analisi, consiste nel ricondurre un input nella classe a cui appartiene il modello, noto alla rete, che, in termini dei parametri a cui la rete stessa è sensibile, gli "somiglia di più".

Un'applicazione interessante di quest'operazione è il riconoscimento di un oggetto o di un'immagine in base ad alcune sue caratteristiche: il cosiddetto **pattern completion**. In questo caso, le immagini che la rete conosce servono a ritrovare quella più simile all'input proposto e restituirla come risposta. Esempi di questo processo potrebbero essere reti neurali che completano immagini disturbate o filtrano rumori di fondo in segnali elettronici o acustici. Per rappresentare con un processo cerebrale umano il corrispettivo fisiologico di quest'operazione, si potrebbe pensare a ciò che il cervello fa quando cerca, per esempio, di ricostruire l'immagine di qualcosa che non sia ben illuminato.

Un altro campo di utilizzo delle potenzialità delle reti neurali artificiali è la ricerca di una configurazione di valori che forniscano la risposta migliore ad un problema dipendente da condizioni vincolanti aggiuntive (problemi di minimo vincolato): si suole indicare questo processo con il termine **constraint satisfaction**. In questo caso, i pesi sono regolati in modo da inibire combinazioni di attivazioni che mal si addicono alla corretta soluzione del problema e possono essere definiti a priori, se non è complesso il criterio per farlo. Questo è l'obiettivo dello studio che verrà descritto in seguito.

Inoltre, una RNA può operare ciò che, a titolo esemplificativo, compie il cervello umano quando associa all'immagine di un oggetto le altre sensazioni che questo oggetto produce: suoni, odori, e così via. In tal caso si parla di **pattern association**, che consiste nel creare una sorta di criterio associativo che colleghi due classi di esempi, in modo che, quando la rete deve cimentarsi con un esempio ignoto della prima classe, restituisca l'elemento della seconda che si adatta meglio ad esso, in base a questo criterio associativo. Anche in questo tipo di rete è necessario un processo di "addestramento", per creare le associazioni che fungono da prototipo per l'interpretazione. Il modo in cui si realizzano le RNA preposte a

quest'ultimo tipo di processo è in genere la topologia combinata di cui si è detto alla fine del paragrafo precedente. Nel caso in cui le due classi di esempi da associare coincidessero, basterebbe una rete ad un unico strato completamente interconnesso. Il lavoro che compie una rete in questo caso è essenzialmente quello di classificare un'immagine confusa, riconducendola in una delle classi note, secondo un processo che si può esprimere con il termine "riconoscimento", da cui la definizione di **pattern recognition** che si usa per quest'ultimo tipo di applicazione.

2.2.4 "Addestramento" delle reti neurali

Si è parlato finora di come la rete neurale riproduca alcune caratteristiche del normale metodo con cui un apparato cerebrale trae conclusioni dai dati ricevuti. Tuttavia, è risaputo che il cervello umano deve le proprie capacità non solo alle caratteristiche fisiche del suo funzionamento, ma anche e soprattutto alle nozioni che esso possiede. Quindi, analogamente, l'ottenimento di una rete neurale artificiale utile ad operare una buona classificazione, deve passare per un "addestramento". "Addestrare" una rete significa, essenzialmente, definire i valori ottimali dei pesi sinaptici per affrontare il problema al quale è preposta.

Nel caso più generale, ciò che si richiede ad una RNA è classificare esempi confusi sulla base di prototipi definiti, quindi essa deve "imparare" tali prototipi. Stabilendo di associare, a determinati input esterni $\mathbf{x}^k$ (espressi in notazione vettoriale perché contengono tutte le attivazioni dello strato di input) altrettanto determinati output $\mathbf{y}^k$, l'addestramento consiste, quindi, nel trovare la configurazione di pesi che consente le giuste associazioni. In base a queste, poi, si può prevedere che, presentando un vettore di cui è ignota la classificazione, la rete neurale lo elabori con un criterio che rispetti la stessa "logica" di base che ha condotto alle associazioni imposte a priori. I metodi di addestramento più diffusi sono essenzialmente due:

1. la regola di Hebb;
2. il metodo della discesa del gradiente.

La regola di Hebb si applica in genere alle topologie completamente connesse. Essa è basata su di un comportamento tipico dei neuroni umani, descritto precedentemente, secondo cui essi tendono a rinforzare la connessione sinaptica se si trovano spesso in stati di uguale attivazione, e ad indebolirla nella situazione opposta. Definendo un parametro η di addestramento, ciò comporta una regola che permette, una volta proposto un esempio, di variare il peso della connessione tra il neurone i-esimo ed il j-esimo come segue:

$$\Delta w_{ij} = \eta \cdot a_i a_j \tag{2.3}$$

in cui il termine di correzione (2.3) dipende dal prodotto delle due attivazioni, sicché tale correzione incrementa o decrementa il peso sinaptico, a seconda che le due attivazioni siano concordi o meno (a prescindere da quali siano effettivamente i segni delle due). Naturalmente, nel caso in cui ci fosse una soglia di attivazione, la (2.3) andrebbe corretta come segue:

$$\Delta w_{ij} = \eta \cdot (a_i - S)(a_j - S) \tag{2.4}$$

Il metodo della discesa del gradiente, che sarà meglio illustrato in seguito, parte da una configurazione casuale dei pesi. In questo caso, ad ogni interpretazione, fatta su esemplari

di cui è nota la classificazione, viene calcolato l'errore commesso in output dal sistema, e si modificano i pesi sinaptici di conseguenza. Così facendo, si produce un movimento di questi valori lungo una traiettoria che porta l'errore verso un minimo, ponendo il sistema in condizione di restituire la risposta più adatta alle previsioni. In sintesi, per descrivere il metodo, si può usare una formulazione vettoriale, indicando con $\mathbf{x}^k$ e $\mathbf{y}^k$, rispettivamente, gli input e output di un esempio da imparare e con $\mathbf{W}$ la matrice dei pesi di connessione fra due strati consecutivi. Nel caso semplice in cui ci sono solo gli strati di input e di output, si valuta il vettore $\mathbf{o}^k = \mathbf{W} \cdot \mathbf{x}^k$, l'output che la rete non addestrata restituisce per l'input $\mathbf{x}^k$, e da qui l'errore:

$$\epsilon = \frac{1}{2}\sum_{i=1}^{N}(y_i^k - o_i^k)^2 = \frac{1}{2}\sum_{i=1}^{N}(y_i^k - \mathbf{W}_i x_i^k)^2 \tag{2.5}$$

Di questo si calcola il gradiente, dipendente da $\mathbf{W}$, che è l'unica quantità variabile della (2.5). Di conseguenza, volendo minimizzare l'errore, si varia ogni peso di una quantità proporzionale a questo gradiente, cambiata di segno, in modo da avvicinarne il valore ad un minimo di questa funzione:

$$\Delta\mathbf{W}_i = -\eta \cdot \nabla_i \epsilon \tag{2.6}$$

In entrambi i sistemi di addestramento bisogna, quindi, fornire, alla rete "grezza", esempi di cui è nota l'interpretazione, in modo che essa adatti a questi i propri pesi sinaptici. Chiaramente, ogni esempio tenderà a modificarli in modo che la rete risponda meglio ad esso in particolare, per cui lavora per uno scopo parziale, ed è per questo che vanno tutti presentati più volte, magari in sequenze differenti. Il fattore η, introdotto con lo scopo di controllare l'entità delle modifiche, ha utilità proprio in quest'ottica. Dopo aver presentato tutti gli esempi, operando le dovute modifiche nelle connessioni, ossia eseguendo quello che si chiama **ciclo di training**, infatti, l'addestramento prosegue con un altro ciclo analogo, in cui η viene ridotto. Il procedimento va proseguito finché l'errore non diventa abbastanza piccolo da permettere di assicurare che la RNA classifichi correttamente tutti gli esemplari.

Tuttavia, anche nel modo di eseguire un ciclo di training, bisogna avere l'accortezza di non insegnare alla rete un campione parziale e/o troppo specifico di un sottoinsieme dell'insieme statistico da analizzare. Il motivo è che, così facendo, la rete subisce un sovrallenamento per certi esempi, con il risultato che basterebbero differenze piccolissime da essi per non ottenerli come risposta, mentre altre classi non ben "imparate" raccolgono elementi fin troppo diversi. Un esempio sarebbe una persona che, esercitandosi a leggere solo con i caratteri di stampa, non riesce poi a capire i simboli di un testo battuto a macchina o scritto a mano.

Nei prossimi capitoli, si passerà ad una breve illustrazione di alcuni modelli di reti neurali più comuni, tra le tante che ne esistono: il percettrone e le sue evoluzioni, e poi le memorie autoassociative, modello usato per l'algoritmo analitico alla base di questo lavoro di tesi.

Capitolo 3

Il percettrone a doppio strato e la sua evoluzione a più strati

3.1 Struttura e funzionamento

Il percettrone, essendo uno dei primi modelli realizzati di dispositivo neurale, possiede naturalmente un particolare valore storico. A parte questo, è un prototipo di RNA dalla struttura abbastanza semplice e costituita da pochi elementi fondamentali. Per questo motivo esso rappresenta un buon esempio da trattare per una discussione introduttiva, che, mantenendosi entro i limiti di un accenno senza la pretesa di approfondire tutti gli aspetti dell'argomento, voglia illustrare in termini pratici il funzionamento di una rete neurale.

Fu Frank Rosenblatt, intorno al 1950, a definire il modello del percettrone. Il suo scopo era, per citarne le parole testuali [Free91]: "illustrare alcune delle proprietà fondamentali dei sistemi intelligenti in generale, senza addentrarsi eccessivamente nelle condizioni particolari, e spesso ignote, che valgono per particolari organismi biologici". In altre parole, lo scopo di Rosenblatt era quello di realizzare un dispositivo che potesse essere sceverato dalle eventuali correlazioni con aspetti specifici della neurofisiologia animale od umana, limitandosi a riprodurre un criterio di trasmissione delle informazioni che somigliasse, entro i limiti di quanto ciò poteva essere realizzabile, a quello dei neuroni fisiologici. Il nome stesso di "percettrone" che fu dato a questo sistema vuole richiamare proprio il processo secondo il quale, partendo da impulsi esterni, il dispositivo riesce a distinguerli e classificarli opportunamente, cosa che avviene senza sostanziali distinzioni per ogni percezione ottenibile dai diversi apparati sensoriali umani ed animali in generale.

Nella sua forma strutturale più generica, un percettrone è costituito da due strati neurali veri e propri, più uno strato preliminare che è posto in relazione con l'esterno, fungendo da dispositivo di percezione e codificazione elettronica. La figura 3.1 mostra un diagramma schematico della struttura del sistema.

Come lo schema rappresenta, si trova all'inizio una zona sensoriale (S), che riceve i segnali dall'esterno (e non deve necessariamente essere presente); i suoi elementi vengono associati ad alcuni dei neuroni della regione successiva, detta *strato associativo* (verranno chiamati in seguito "neuroni A") secondo un assegnamento equilibrato ma casuale tanto nella scelta dell'unità A che riceve ogni segnale quanto nel valore e segno del peso sinaptico associato al canale di ricezione. Le unità A vengono poi collegate ad alcuni tra i neuroni di risposta ("neuroni R"), così chiamati perché, per l'appunto, dalle loro attivazioni finali si

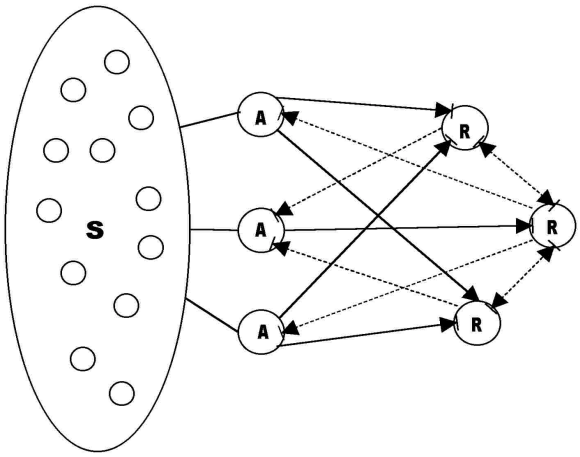

Figura 3.1: Schema strutturale del percettrone. Entro l'ellisse S si trova la regione percettiva, e le A e le R distinguono i neuroni associativi da quelli di risposta. I legami dalla zona S alla zona A non sono stati disegnati tutti per non appesantire la figura. Le connessioni tra A ed R sono eccitative se rappresentate da una linea continua ed inibitorie se rappresentate da linee tratteggiate, e si dirigono nel senso delle frecce.

estrae la risposta del percettrone all'input fornitogli. I collegamenti A–R non sono casuali, ma seguono una logica che dipende dal problema effettivo da risolvere. Inoltre, tutti i neuroni A che non inviano un segnale ad un certo neurone R, ne ricevono uno inibitorio da esso, di modo che quando questo si attiva, tende a spegnere le unità A che non hanno contribuito alla sua attivazione. Secondo la stessa logica vengono imposte connessioni inibitorie anche tra i neuroni R. Il motivo di questa costruzione si comprende pensando che i neuroni utilizzati nel percettrone sono del tipo binario (0/1 o $-1/1$) e a soglia. Con questa configurazione sinaptica, quindi, si crea una competizione tra i neuroni R, per cui si prevede che il *vettore di output* finale del percettrone (il modo in cui si indica l'insieme di tutte le attivazioni dello strato R) abbia una sola componente uguale ad 1 e tutte le altre uguali a 0 o -1.

Quanto detto permette di comprendere facilmente come il funzionamento di un sistema di questo genere segua da vicino le regole del modello di McCulloch-Pitts illustrate nel capitolo precedente, se le si immagina espresse in termini di variabili binarie e logica booleana.

3.2 Utilità ed addestramento

È chiaro come il percettrone sia essenzialmente un dispositivo adatto alla classificazione di elementi in base ad una qualche loro caratteristica. Con questo obiettivo, i problemi da porsi sono due:

- se sia possibile, dato un preciso problema di classificazione, creare un percettrone capace di risolverlo;
- come si fa ad addestrare il dispositivo per metterlo in grado di funzionare come serve all'operatore.

La risposta alla prima domanda conduce alla discussione circa la *separabilità lineare* delle classi di esempi, che è una maniera rigorosa di esprimere matematicamente le con-

dizioni in cui un percettrone riesce a funzionare correttamente. Per poter discutere questo argomento, però, bisogna prima illustrare il metodo che va seguito per addestrare il sistema. Noto questo procedimento, verrà da sé la condizione essenziale perché il problema sia risolubile mediante un dispositivo di questo genere.

In base a quanto detto finora, è abbastanza chiaro che addestrare il percettrone significa dare ai pesi delle connessioni i valori giusti perché in questi valori risiede, è importante sottolinearlo ancora una volta, la "logica" con cui esso opera la classificazione. Il criterio da seguire per determinare questi valori consiste nell'impostarli inizialmente a caso e poi, dopo aver provato a classificare alcuni esempi noti, correggerli volta per volta, finché la rete non smette di sbagliare. La forma semplice in cui viene codificata la risposta, che si è detto consiste in un vettore di output con un solo elemento diverso da zero, permette di impostare un processo di aggiustamento abbastanza semplice [Free91]. Il criterio è il seguente: dopo aver fatto elaborare un esempio al percettrone, forzando i neuroni R a rappresentare il vettore di output previsto, si rafforzano i pesi delle connessioni A–R tra elementi con attivazione uguale e si indeboliscono quelli tra elementi con attivazione diversa, tenendo conto che per l'attivazione dei neuroni A si usa il risultato dell'elaborazione dell'esempio, e non per forza deve corrispondere, inizialmente, all'output previsto. In questo modo succede in pratica che, se la risposta all'input presentato è il neurone R giusto, si rafforzano tutti i pesi sinaptici A–R tra le unità A che si sono attivate e l'unità R che alla fine si accende. Se, invece, l'unità R accesa è quella sbagliata, gli stessi pesi sinaptici vengono indeboliti. Esiste un teorema [Bish95], in cui si dimostra rigorosamente come questo processo di addestramento conduca, in un numero *finito* di prove, all'ottenimento dei valori ottimali di tutti i pesi (per la sua dimostrazione, si rimanda all'appendice C della tesi). L'unica condizione è proprio quella di avere un problema *linearmente separabile*. Si rende necessaria, a questo punto, la comprensione di tale definizione.

Si pensi all'esempio semplice in cui il percettrone è costituito da un solo neurone R e due neuroni A, senza strato sensoriale. Si ipotizzi inoltre che i tre neuroni siano a risposta binaria, ma del tipo $(1/0)$, anziché $(-1/1)$. Naturalmente, chiamando rispettivamente x_1, x_2, w_1 e w_2 le attivazioni ed i pesi sinaptici con l'unità R dei due neuroni A, si possono avere quattro diverse configurazioni di input, ordinando i neuroni secondo gli indici: $(0,0)$, $(0,1)$, $(1,0)$ e $(1,1)$. Rappresentando graficamente queste configurazioni si ottengono quattro punti di un piano i cui assi sono x_1 e x_2. Dato che l'output è un solo neurone, possono esistere al massimo due classi, contraddistinte dai due possibili stati di attivazione del neurone R, che si ottengono con questa legge:

$$\begin{cases} 1 & \text{se} \quad w_1 x_1 + w_2 x_2 > S \\ 0 & \text{se} \quad w_1 x_1 + w_2 x_2 \leq S \end{cases} \tag{3.1}$$

in cui S è la soglia dell'unità R. Se nel piano $x_1 x_2$ si traccia la retta di equazione $S = w_1 x_1 + w_2 x_2$, si divide questo piano in due regioni, caratterizzate ciascuna dal fatto di contenere tutti i punti corrispondenti ad input che vengono classificati nello stesso modo. Questo fatto viene espresso dicendo che questa retta è la *frontiera di discriminazione* delle due classi che, in questo caso, è lineare. Si capisce come, giostrando con i valori di S e dei pesi, si possano trovare quelli migliori per operare la distinzione corretta. La figura 3.2 mostra graficamente come si possano definire così le frontiere di discriminazione per eseguire con un percettrone le operazioni di AND e OR binarie.

Il fatto che le operazioni di AND e OR possano essere definite da una retta di discriminazione, si indica affermando che esse rappresentano problemi *linearmente separabili*.

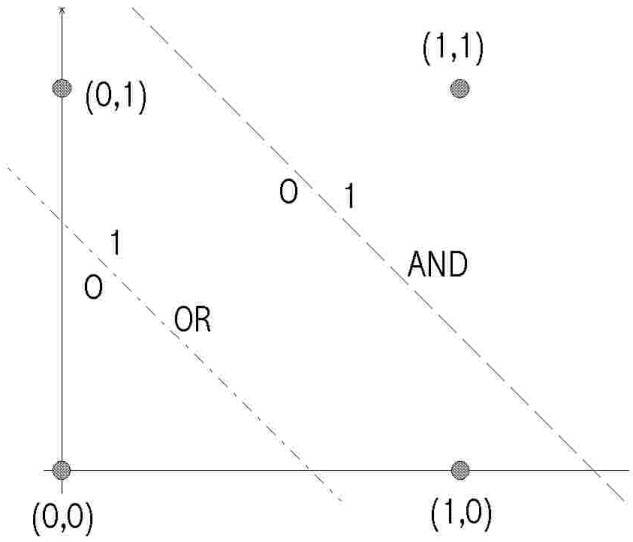

Figura 3.2: Frontiere di discriminazione per le operazioni di AND e OR con i valori di classificazione dei due gruppi di input di ogni classe.

Si ponga, adesso, di voler risolvere un altro problema. Osservando la figura, come dovrebbe essere disegnata la frontiera di discriminazione per compiere l'operazione XOR, che assegna 1 a $(1, 0)$ e $(0, 1)$ e 0 a $(1, 1)$ e $(0, 0)$? In questo caso, la frontiera è data dall'insieme di entrambe le linee AND e OR, quindi il problema non è linearmente separabile. Il fatto che non si riesca, in questo caso, a trovare una retta che separa il piano in due zone di classificazione continue e distinte, ricordando il significato dei coefficienti di questa retta, significa che non è possibile trovare una terna di valori S, w_1 e w_2 che permettano ad un percettrone fatto in questo modo di classificare il problema.

Generalizzando l'esempio, il *teorema di convergenza del percettrone*, citato in precedenza, mostra che un percettrone non può mai essere addestrato a risolvere problemi di classificazione non linearmente separabili. Naturalmente, nel caso generale, ciò che qui è una retta, diventa un iperpiano nello spazio ad n dimensioni (un'equazione lineare in n variabili), se n sono i neuroni di input. È possibile vedere [Free91] come, se si aggiunge, tra lo strato A di input e il neurone R di output, uno strato H nascosto di due neuroni, uno con soglia minore di 1 e l'altro maggiore, il problema XOR divenga risolubile, ma questo modello di RNA non è più un percettrone, bensì una sua evoluzione più complessa, chiamata **Multi-Layer Perceptron**.

3.2.1 Aspetti implementativi

È il momento di introdurre, adesso, un argomento di discussione necessario in ogni campo di applicazione delle reti neurali. È chiaro che non è facile progettarne una alla cieca, senza aver valutato le sue prestazioni. Tale stima, infatti, rimane in gran parte empirica, per quanto esistano riferimenti su cui basarsi per stabilire, ad esempio, quanti strati nascosti possano servire e quanti neuroni ci vogliano per ciascuno, a parte quelli di input e output,

il cui numero e specie sono definiti dal problema e dal criterio con cui lo si pone in termini informatici.

Ciò che aiuta in questo compito, è la simulazione al computer: ciò che serve è un programma che crei una sorta di "rete neurale virtuale", il cui comportamento permetta di testare le prestazioni del modello usato, prima di realizzarlo in pratica, il che potrebbe condurre a spese ingenti senza vantaggi oggettivi. Naturalmente, i tempi di calcolo potrebbero essere grandi e così anche i requisiti di memoria, dato che il computer ha uno spazio limitato di RAM in cui depositare i valori necessari ed una sola unità di elaborazione, il suo processore, per cui esso può simulare i risultati della rete, ma non certo riprodurne il calcolo parallelo. È comunque un buon mezzo per fare le prove, a scapito di un poco di attesa. Per questo, è utile discutere brevemente una maniera generica di implementare un programma simile.

Dal momento che questo non vuol essere un trattato di applicazioni informatiche specifiche ma solo un'idea per capire come potrebbe essere posto il problema, verrà proposto un caso semplice: il percettrone ad un solo output. Qui, i dati di interesse sono, per i neuroni A: il peso sinaptico di collegamento con l'unica unità R presente e l'attivazione; è necessario, inoltre, conoscere la soglia di R. Non serve una variabile in cui porre l'output, perché lo si utilizza direttamente dalla procedura che lo fornisce, essendo unico, e non serve nemmeno conoscere le soglie dei neuroni A, visto che le loro attivazioni sono fissate a priori. In ogni caso, avendo a che fare con neuroni a risposta binaria, si possono rappresentarne le attivazioni con variabili booleane, contemplate in molti linguaggi di programmazione. È comodo usare un modello, come si usa dire, orientato agli oggetti, che snellisce di molto la realizzazione del codice, che qui sarà descritta in una forma non legata particolarmente ad alcun linguaggio, ma basata piuttosto su un metalinguaggio le cui specifiche sono descritte nell'appendice A. Inoltre, con questo criterio, è facile associare ad ogni elemento i valori opportuni con il loro significato. Ecco una possibile struttura di dati per un oggetto "neurone":

```
STRUCTURE Neuron {
   BOOL Activation;
   REAL Weight;
}
```

Questa struttura, che serve solo per i neuroni A, contiene l'attivazione, del tipo "vero/falso" ed il peso sinaptico con l'output. Le variabili relative all'unità di output possono essere dichiarate singolarmente, essendo una per ogni campo. Per calcolare l'input del neurone R, nello schema ad attivazioni binarie, è chiaro che la somma pesata delle attivazioni si riduce a quella dei pesi sinaptici di collegamento tra quest'unità e tutte le unità A accese. Da qui, per calcolare l'attivazione, basta porre il confronto dell'input ricevuto con la soglia assegnando "vero" o "falso" in base a quale dei due valori è maggiore dell'altro.

Immaginando un programma che simuli un percettrone con N elementi A, c'è bisogno, quindi, di una variabile globale reale, che si chiamerà `Output_Threshold`, rappresentante la soglia del neurone di output e di un array di N elementi del tipo `Neuron`, che sarà denominato `In_Unit[]`. La funzione di aggiornamento dell'output, supposto di aver fissato gli input, può essere la seguente:

```
FUNC BOOL Update()
   REAL Input = 0.0;
   INT i;
   FOR (i, 1, N)
      IF In_Unit[i].Activation = TRUE
```

```
            Input = Input + In_Unit[i].Weight;
         END IF
      END FOR
      IF Input > Output_Threshhold
         RETURN true;
      ELSE
         RETURN false;
      END IF
   END FUNC
```

Eseguire il programma comporta una fase di richiesta ed impostazione dell'input, ed in seguito l'applicazione di questa funzione.

Naturalmente, il programma deve provvedere anche ad addestrare il percettrone virtuale, seguendo la regola di Rosenblatt: rinforzare le connessioni fra neuroni che hanno contribuito ad una risposta corretta, ed indebolire quelle tra unità che hanno condotto ad una errata. A questo scopo, si può definire un parametro reale, diciamo `Delta`, che va aggiunto al peso dei neuroni A nel primo caso e sottratto nel secondo caso. Si può procedere nel seguente modo: si propongono al percettrone diversi esempi di addestramento e si fanno correzioni di una quantità uguale a `Delta`, dopo di che si diminuisce quest'ultimo e si ripropongono gli esempi, finché il percettrone non classifica tutti gli esempi correttamente. Naturalmente, gli esempi saranno array di valori booleani, da associare a ciascun neurone A simulato, la qual cosa non viene espressa per esteso perché è banale, e la si indica con una funzione chiamata `Initialize(Example[i])` in cui `Example[i]` è l'i-esimo esempio di addestramento a cui corrisponde l'output previsto `Output[i]`. Ecco un esempio per la funzione[1]:

```
SUB Train(REAL Delta,REAL Decr)
   BOOL Result;
   INT Errors = 1, i, j;
   DO WHILE Errors > 0
      Errors = 0;
      FOR (i, 1, Examples_Num)
         Initialize(Example[i]);
         Result = Update();
         IF NOT (Result = Output[i])
            Errors = Errors + 1;
            FOR (j, 1, N)
               IF (In_Unit[j].Activation = Output_Activation)
                  In_Unit[j].Weight -= Delta;
               END IF
            END FOR
         ELSE
            FOR (j, 1, N)
               IF In_Unit[j].Activation = Output_Activation
                  In_Unit[j].Weight += Delta;
               END IF
            END FOR
         END IF
      END FOR
      Delta = Delta - Decr;
```

[1]Per abbreviare si intenda `X += A` come `X = X+A`

```
    END DO;
END SUB
```

In questa subroutine, quando il risultato dell'aggiornamento è giusto, un'iterazione su tutti i neuroni di input incrementa il peso di tutti quelli la cui attivazione è uguale all'output; nel caso di un errore, invece, decrementa i pesi delle medesime connessioni della stessa quantità. Se il numero di errori è maggiore di zero, viene decrementato il fattore di correzione di una quantità stabilita dal programma e si ripete il ciclo, altrimenti la subroutine si chiude, terminando il processo di addestramento. Essa, comunque, nel modo semplice in cui è stata proposta, non considera la possibilità che il fattore `Delta` possa ridursi a valori minori di zero, senza che la rete smetta di sbagliare. A questo problema, in realtà, si dovrebbe ovviare proponendo abbastanza esempi, o valutando la possibilità di fare restituire alla subroutine un valore, ad esempio "vero" se la rete smette di sbagliare prima che `Delta` scenda sotto lo zero, o "falso" se avviene il contrario. In quest'ultimo caso, sarà necessario valutare se, ed eventualmente con quali altri esempi, bisogna ulteriormente addestrare la struttura.

3.3 Back-propagation network

Questo modello di rete, evoluzione del precedente, è il prototipo della rete a strati. Il fatto che venga addestrato con il metodo della retro-propagazione — o "discesa" — del gradiente degli errori, giustifica la sua denominazione. Questa rete neurale è strutturata a strati, almeno tre, di cui uno di input, uno di output ed uno nascosto. Come si è detto nel caso del problema XOR, descritto in precedenza, l'aggiunta di strati nascosti di unità può mettere la rete in grado di risolvere problemi di classificazione caratterizzati da frontiere ben più complesse di quelle lineari.

La figura 3.3 mostra un'immagine schematica di back-propagation network (BPN). Come è tipico delle reti a strati, essa contiene soltanto connessioni tra neuroni di uno strato con neuroni del successivo. È chiaro che questa rete, quindi, elabori le informazioni semplicemente trasferendo i dati dallo strato di input a quello di output, processo che passa per i pesi delle connessioni che i dati debbono attraversare. Questi pesi, quindi, sono ciò che stabilisce, alla fine, che input riceve ogni neurone nascosto prima, e di output poi, e quindi la risposta della rete. Per chiarire matematicamente il processo, si supponga che i neuroni di input siano organizzati, usando la notazione vettoriale, secondo un ordine preciso, sicché sia possibile indicare ciascuno con un indice intero, e con un vettore $\mathbf{I}$ l'insieme delle attivazioni. Naturalmente, in questo caso, se ci sono l neuroni di input, m neuroni nascosti e n neuroni di output, queste saranno le dimensioni dei vettori di attivazione dei tre strati. Invece, si potranno rappresentare i pesi sinaptici in forma di matrici, che sono due distinte: una, chiamata $\mathbf{W}^{ih}$, per le connessioni input-hidden, e l'altra, indicata con $\mathbf{W}^{ho}$, per le connessioni hidden-output. Naturalmente, la prima matrice è $l \times m$ e l'altra è $m \times n$. Con questa notazione, si può scrivere l'espressione del vettore d'attivazione dell'hidden layer come prodotto matriciale:

$$\mathbf{H} = F(\mathbf{W}^{ih} \cdot \mathbf{I}) + \Theta^h \tag{3.2}$$

in cui, con la F, si indica la funzione d'attivazione (che può avere forme diverse secondo i diversi modelli di BPN) nel senso che, per ogni elemento nascosto, si applica al corrispondente elemento del vettore che si ottiene dal prodotto matrice-vettore in argomento, con

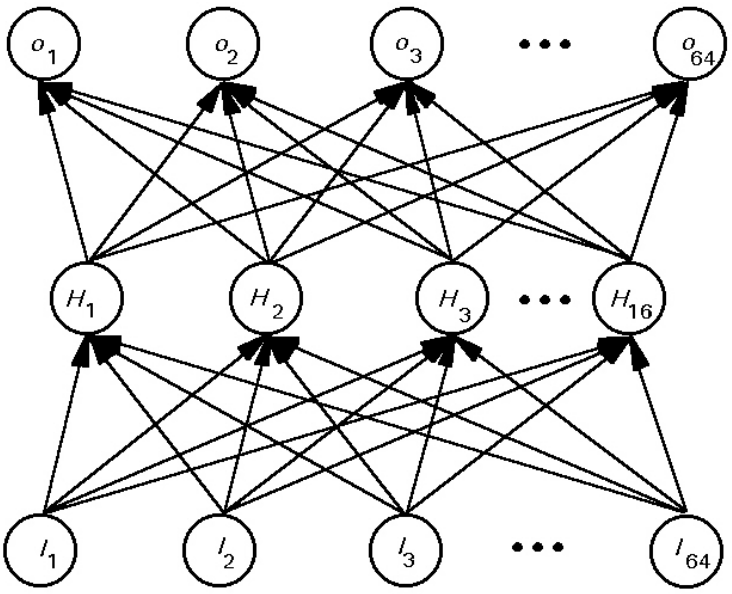

Figura 3.3: Modello di BPN, con uno strato di neuroni di input (in basso, indicati con "I"), seguito dallo strato nascosto ("H") e da quello di output ("O"). Le linee rappresentano le connessioni sinaptiche, che sono a senso unico, da uno strato all'altro. Si noti l'assenza di legami tra unità I ed unità O, o tra elementi dello stesso strato.

la condizione però che la F sia la stessa per tutte le unità. Il termine Θ^h rappresenta un vettore di valori di soglia associati a ciascun neurone. Allo stesso modo si determina il vettore d'attivazione dell'output layer:

$$\mathbf{O} = F(\mathbf{W}^{ho} \cdot \mathbf{H}) + \Theta^o \tag{3.3}$$

L'applicazione della rete BPN è molto utile in tutti quei casi nei quali sia difficile ricondurre un problema ad una precisa funzione delle variabili in ingresso, per cui è più utile una stima per associazioni che un calcolo deterministico. È ciò che potrebbe capitare, ad esempio, per quei dati che dipendono da troppe variabili, o per i casi in cui la funzione che li correla per fornire la risposta è troppo complicata. Altri casi in cui si presenta questo problema sono quelli in cui, ad un certo punto dell'elaborazione, bisogna compiere la scelta migliore tra le possibili opzioni, il che un computer, come detto sopra, non riesce a fare, a meno che un programma non gli indichi i parametri in base ai quali stabilire cosa è meglio di cos'altro, il che non sempre è facile da implementare in algoritmi.

3.3.1 Addestramento

Il criterio che si usa per addestrare la rete BPN è, si è detto, quello che le da il nome, ossia la retro-propagazione del gradiente di errore.

Date un certo numero di coppie, diciamo N, di esempi, costituite ciascuna da un vettore di input ed uno di output, ciò che si fa è proporre l'input di ciascuno alla rete, ed osservare l'errore che essa commette in output, e poi correggere i pesi sinaptici di una quantità proporzionale a quest'errore, cambiata di segno per condurre alla regione di errore minimo. Seguendo più da vicino il processo [Free91, Kot93], si parte dalla coppia di vettori $\mathbf{I}^k$ e $\mathbf{O}^k$, che costituiscono il k-esimo esempio, e si pone che sia $\mathbf{Y}^k$ l'output "errato" della rete non addestrata. È possibile, in questo caso, calcolare un errore, che è essenzialmente funzione

dei pesi sinaptici:

$$E_k = \frac{1}{2}\sum_i (Y_i^k - O_i^k)^2 \tag{3.4}$$

dove i rappresenta un indice che scorre su tutti i neuroni di output.

Quello che bisogna stabilire è in che direzione cambiare questi pesi, tenendo conto dell'errore. Questo processo viene fatto in tanti stadi quante sono le serie di connessioni, quindi il numero degli strati meno uno. Trattando l'esempio semplice di una rete a tre strati, quindi, i passaggi sono due: prima l'aggiornamento dei pesi hidden-output, e poi quello dei pesi input-hidden. Se l'errore è funzione dei pesi sinaptici, si può pensare che il suo gradiente è associato con la variazione dell'errore stesso in funzione di essi, per cui, se si varia ogni termine di una quantità proporzionale alla componente corrispondente del gradiente di E rispetto ai pesi cambiato di segno, questo passaggio tende a spostare E verso un minimo, e quindi la rete verso una configurazione ottimale dei pesi. A questo punto, serve conoscere le derivate parziali di questo errore rispetto ad un generico peso:

$$\frac{\partial E_k}{\partial W_{ij}^{ho}} = -(Y_j^k - O_j^k)\frac{\partial F}{\partial net_j^{ok}}\frac{\partial net_j^{ok}}{\partial W_{ij}^{ho}} \tag{3.5}$$

dove gli indici hanno i seguenti significati: k indica l'esempio, i l'unità nascosta da cui parte il peso e j l'unità di output cui esso giunge; net_j^{ok} è l'input complessivo che il neurone j-esimo riceve dallo strato nascosto, per il k-esimo esempio. Si è scritta la forma in questo modo perché esplicitamente, la funzione F ha proprio net_j^{ok} come argomento, e quanto alla derivata di net_j^{ok} rispetto al peso, la si deduce immediatamente:

$$net_j^{ok} = \sum_i W_{ij}^{ho} H_i^k + \Theta_j^o \Rightarrow \frac{\partial net_j^{ok}}{\partial W_{ij}^{ho}} = H_i^k \tag{3.6}$$

e, da essa, la variazione prevista nel peso W_{ij}^{ho}:

$$\Delta W_{ij}^{ho} = \eta \cdot (Y_j^k - O_j^k) H_i^k \frac{\partial F(net_j^{ok})}{\partial net_j^{ok}} \tag{3.7}$$

in cui il parametro η rappresenta una quantità che controlla l'entità della variazione, e bisogna che non sia troppo grande, per evitare il rischio di sovra-addestrare la rete per un certo esempio, impedendole di stabilizzarsi correttamente.

Una nota finale, non ultima in importanza, è che la funzione d'attivazione deve essere differenziabile, altrimenti il processo non si può sviluppare. Questo non è un problema per le attivazioni lineari, ma lo diviene quando vorremmo reti BPN a risposta binaria, il che non è del tutto fuori discussione: in questo caso, si può adoperare la funzione *logistica*:

$$F(x) = \frac{1}{1 + e^{-\lambda x}}$$

Questa funzione, ha una forma sigmoidale, che è tanto più vicina al gradino quanto più il coefficiente λ è grande. In questo caso, si può procedere all'aggiornamento ed all'interpretazione di un esempio, e poi considerare "accesi" tutti i neuroni che avranno un valore di attivazione compreso fra un limite inferiore positivo stabilito ed 1.

Per l'aggiornamento dei pesi dello strato nascosto, si opera nello stesso identico modo, però sorge il problema che non si conosce quale dovrebbe essere il corretto vettore d'attivazione dello strato nascosto, perché l'esempio non contempla la conoscenza di questo.

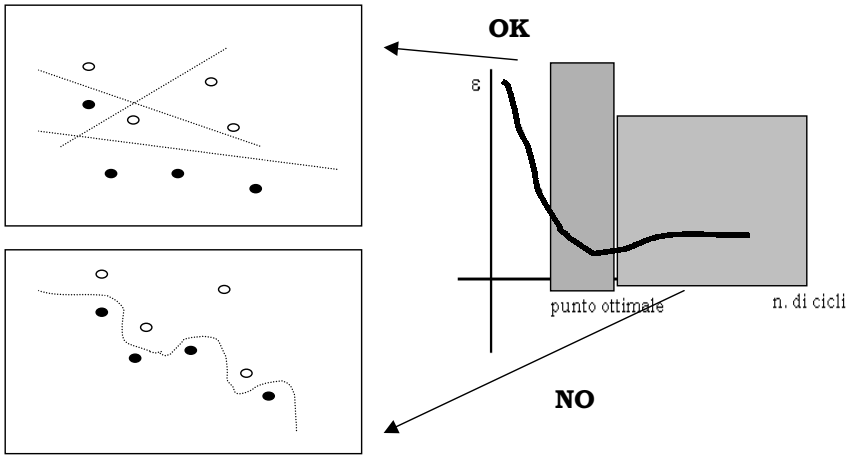

Figura 3.4: Un esempio di andamento dell'errore della risposta di una BPN in un addestramento. Si noti come, dopo un abbassamento dell'errore ad un minimo ottimale, esso torni a salire se si sovra-addestra la rete neurale, producendo frontiere di discriminazione non più rettilinee, il che, pensando a come ogni neurone riceve il suo input, è evidentemente poco auspicabile. Questa fase, come spiegato nel testo, comporta l'ottenimento di una RNA troppo specifica nella classificazione di alcuni esempi e troppo poco esercitata a riconoscerne altri diversi, anche se appartenenti allo stesso campione statistico.

Quindi, bisogna riprendere l'errore di output della (3.4), e considerare la sua dipendenza dai pesi tra input layer e hidden layer. Indicando con l l'indice dell'unità di input, questa dipendenza è chiara dalla seguente derivata parziale, scritta di seguito:

$$\frac{\partial E_k}{\partial W_{ij}^{ih}} = -\sum_j (Y_j^k - O_j^k) \frac{\partial O_j^k}{\partial net_j^{ok}} \frac{\partial net_j^{ok}}{\partial H_i^k} \frac{\partial H_i^k}{\partial net_i^{hk}} \frac{\partial net_i^{hk}}{\partial W_l^{ih} i} \tag{3.8}$$

Da quest'espressione complicata, si trae la regola d'aggiornamento considerando come alcune variabili si derivino dalle altre in base alla regola con cui transitano le informazioni attraverso la rete:

$$\Delta W_{li}^{ih} = \eta \cdot \frac{\partial F(net_i^{hk})}{\partial net_i^{hk}} I_l^k \sum_j (Y_j^k - o_j^k) \frac{\partial F(net_j^{ok})}{\partial net_j^{ok}} \tag{3.9}$$

È bene spendere qualche parola circa il problema più importante, che sta alla base dell'addestramento: la scelta degli esempi. In realtà non esistono regole per scegliere il set di vettori d'addestramento, però è bene fare una scelta oculata e soddisfacente nel contempo, perché è importante che una rete abbia avuto un addestramento che contempli esempi generici di tutte le sottoclassi dell'insieme statistico da cui sono stati presi, altrimenti si rischiano risposte poco significative per alcuni esempi. Se poi i vettori di addestramento sono troppo pochi, la rete non imparerà abbastanza, mentre se sono troppi, la rete potrebbe non riuscire ad ottimizzare i pesi in modo tale da fornire una risposta estremamente confacentesi a tutti. Se poi si esagera, in ogni caso, con le correzioni per adattare la rete ad un determinato set di vettori, si potrebbe rischiare di deformare la frontiera di discriminazione del problema, e costruire una rete con una logica sbagliata. Per fare un esempio, è quello che potrebbe succedere ad una persona che, addestrata a riconoscere una determinata razza di cane, non riesce più a identificare come "cane" un esemplare di qualsiasi altra razza.

Altro problema che non ha una soluzione universale è quello delle dimensioni della rete. Se infatti è evidente, caso per caso, quanti neuroni di input e di output servano,

non è altrettanto ovvio quanti strati intermedi possano servire per ottenere una risposta soddisfacente, né quanti neuroni ci vogliano in ciascuno degli strati. Quello che aiuta è, fondamentalmente, l'esperienza e le prove ripetute con quantità diverse, tenendo conto che già, con un solo hidden layer, le capacità della rete crescono di una quantità notevole.

3.3.2 Aspetti implementativi

Nel caso di una rete BPN, bisogna considerare che la RAM viene messa alla prova in modo consistente, dato che il numero di neuroni è abbastanza grande. La prima cosa da fare è creare le strutture di dati che servono. Contrariamente a quanto si potrebbe pensare, in questo caso è più comodo creare una struttura di dati per lo strato di neuroni, memorizzando le caratteristiche di ciascuno come array di valori reali:

```
STRUCTURE Layer_Type {
   INT Count;
   REAL[] Act;
   REAL[][] Weight;
}
```

In questo esempio, la struttura `Layer_Type` contiene un array monodimensionale di valori reali, che sono le attivazioni, ed uno a due dimensioni, che rappresenta i pesi sinaptici con lo strato precedente (il primo indice si riferisce all'elemento di questo strato e il secondo all'elemento collegato dell'altro strato). Praticamente, la struttura tiene conto dei pesi sinaptici con lo strato che invia ad essa il proprio output. C'è anche un intero che contiene il numero di elementi dello strato. Un elemento fondamentale, qui, è un riferimento ad una struttura dello stesso tipo, che corrisponde allo strato precedente. Si può, allora, immaginare di avere un array di strutture `Layer_Type`, per cui ognuna considera come suo precedente l'elemento dell'array con l'indice inferiore. Siccome tale soluzione è più semplice da implementare e, spesso, comporta uno stress minore per la memoria del computer, sarà utilizzata questa. Per aggiornare la rete, allora, bisogna, dopo aver inizializzato lo strato iniziale — e si supporrà di averlo fatto in quanto è una procedura banale — aggiornare gli strati in ordine, ottenendo come risultato il vettore di attivazione dell'ultimo strato. Per strutturare una procedura di aggiornamento con questo criterio e questo tipo di struttura dati, si porrà di aver definito nel programma un array di nome `Layer[]`, di `n_Layer` elementi numerati da 0 a $n\_Layer - 1$, di modo che `Layer[0]` sia l'input layer; si porrà inoltre di avere neuroni ad attivazione logistica, in cui con `EXP()` si indica la funzione esponenziale:

```
SUB Update()
   INT i, j, k;
   REAL Input;
   FOR (i, 1, n_Layer - 1)
      FOR (j, 0, Layer[i].Count)
         Input = 0 ;
         FOR (k, 0, Layer[i-1].Count)
            Input += Layer[i].Weight[j][k] * Layer[i-1].Act[k];
         END FOR
         Layer[i].Act[j] = 1 / (1 + EXP(Input * Coeff));
      END FOR
   END FOR
END SUB
```

Questa subroutine, in pratica, scorre, per ogni strato, tutti i neuroni del precedente, in modo da calcolare, per ogni proprio elemento, l'input totale, a cui viene poi applicata la funzione logistica, in cui il coefficiente è espresso dalla variabile `Coeff` presente nel listato. In questo modo, viene prodotto il vettore di attivazione dello strato. Fatto questo, la subroutine passa allo strato successivo, fino ad arrivare all'ultimo. Poi, banalmente, si preleverà da questo il vettore di attivazione per ottenere la risposta. Adesso è il momento di implementare una funzione per l'addestramento. Ancora una volta si porrà di indicare con la subroutine `Initialize(Example[i])` l'inizializzazione dell'input layer al vettore d'attivazione dell'esempio i-esimo, cui corrisponderà l'output `Output[i]`. Prima di procedere, è il caso di riscrivere le espressioni 3.7 e 3.9 nel caso in cui F sia la funzione logistica, la quale ha derivata:

$$F'(x) = \frac{\lambda}{2(\cosh \lambda x + 1)}$$

per cui si ottengono le seguenti espressioni:

$$\Delta W_{ij}^{ho} = \eta \cdot (Y_j^k - O_j^k) H_i^k \frac{\lambda}{2[\cosh(\lambda net_j^{ok}) + 1]} \tag{3.10}$$

$$\Delta W_{li}^{ih} = \eta \cdot \frac{\lambda}{2[\cosh(\lambda net_j^{hk}) + 1]} I_l^k (Y_j^k - O_j^k) \frac{\lambda}{2[\cosh(\lambda net_j^{ok}) + 1]} \tag{3.11}$$

Ecco la subroutine di addestramento, nel caso di BPN a tre strati:

```
SUB Train(Real Eta)
   INT i, j, k, l;
   REAL x, y, z;
   FOR (k, 1, n_Examples)
      Initialize(Example[k]);
      Update();
      FOR (j, 0, Layer[2].Count)
         z = Layer[2].Act[j] - Output[j];
         x = Layer[2].Act[j];
         x = LOG(x / (1-x)) / Coeff;
         FOR (i, 0, Layer[1].Count)
            y = Eta * z * Layer[1].Act[i];
            Layer[2].Weight[j][i] -= y*Coeff /(2*COSH(Coeff*x)+1);
         END FOR
      END FOR
      FOR (j, 0, Layer[1].Count)
         FOR (i, 0, Layer[2].Count)
            x = Layer[2].Act[i];
            z += (Layer[2].Act[j]-Output[j])*LOG(x/(1-x))/Coeff;
         END FOR
         x = Layer[1].Act[j];
         x = LOG(x / (1-x)) / Coeff;
         FOR (i, 0, Layer[0].Count)
            y = Eta * z * Layer[0].Act[i];
            Layer[2].Weight[j][i] -= y*Coeff /(2*COSH(Coeff*x)+1);
         END FOR
      END FOR
   END FOR
END SUB
```

Come si nota, il funzionamento è simile per l'addestramento dello strato 2 (output) e 1 (hidden), tranne che per il fatto che il coefficiente indicato con z, che contiene la dipendenza dall'errore, nel primo caso dipende dall'errore di un singolo neurone di output, ossia quello in esame, mentre nel secondo caso, dipende invece dall'errore di tutti i neuroni di output. Si noti che, invece di calcolare banalmente l'input di un neurone nascosto o di output, si è applicata, in modo più semplice e senza dubbio più rapido, l'inversa dalla funzione logistica:

$$y = \frac{1}{1 + e^{-\lambda x}} \Rightarrow x = \frac{1}{\lambda} \ln \frac{y}{1 - y}$$

che è sempre sensata, perché y è compreso tra 0 ed 1, esclusi entrambi, e soprattutto è una funzione ben definita, essendo monotona la funzione logistica. Lo schema delle RNA a strati risulta un po' laborioso, come si vede, nel calcolo delle correzioni dovute

all'addestramento, che sono piene di indici e quindi rischiano di condurre ad una certa confusione, ma in realtà sono abbastanza semplici. Per questo si è presentato qui questo esempio. Un altro elemento di comprensibilità è il fatto che queste RNA operano in modo lineare la loro interpretazione, che semplicemente "passa" per tutti gli strati, arrivando all'ultimo, che restituisce la risposta. Adesso si passerà, nel prossimo capitolo, ad un altro tipo di reti neurali, che si addentrano nel campo specifico di analisi cui la presente tesi è dedicata, e vi si noteranno alcuni elementi concettualmente più semplici, ma anche alcune complicazioni nello schema complessivo dei processi da mettere in funzione, almeno rispetto allo schema fluido delle BPN e del percettrone.

Capitolo 4

Reti neurali artificiali per la soluzione dei problemi di ottimizzazione

L'esempio dei sistemi neurali realistici, proposti nel capitolo precedente, è servito ad illustrare in che modo interagiscono, fisicamente, i componenti elementari di una RNA e come, da tale interazione, l'intero sistema perviene all'interpretazione dell'input ricevuto. Volendo andare oltre l'esempio, si comprende che, in generale, qualsiasi RNA funziona secondo un processo analogo: ogni neurone riceve le attivazioni di altri, le somma e genera un segnale a partire da tale somma. È chiaro, da ciò, che sono due gli elementi caratterizzanti di un modello specifico di rete neurale artificiale: lo schema topologico delle connessioni sinaptiche e la funzione di attivazione dei neuroni.

Date queste premesse, definire un modello di rete neurale adatto ad affrontare una determinata questione, corrisponde a specificarne tali elementi. Dal momento che la presente tesi intende realizzare un'applicazione specifica di calcolo neurale per risolvere un definito problema di fisica, la cosa da fare è, in ultima analisi, stabilire lo schema topologico e la funzione d'attivazione neurale più adatte allo scopo. Il primo passo verso quest'obiettivo è quello di stabilire in quale classe generale di problemi (vedi capitolo 2) si inserisce quello in esame. Ciò conduce già ad una selezione tra i numerosi modelli esistenti di RNA utilizzabili. In un secondo momento, si rende necessario uno sguardo più ravvicinato alla questione specifica da risolvere, il che produce una serie di aggiustamenti che definiscono o modificano i vari aspetti del modello iniziale, conducendo finalmente a quello da cui si è preso spunto per l'algoritmo che la presente tesi ha lo scopo di proporre.

I sistemi che interessa esaminare sono quelli dedicati all'**ottimizzazione**. Nel campo dell'informatica, si definisce con questo termine l'operazione che ha lo scopo di trovare, tra le possibili soluzioni di un problema, quella che si accorda meglio a determinate condizioni al contorno. Lo stesso concetto, in termini matematici, è ciò che si denomina *problema di minimo vincolato*. In realtà, prima di poter giungere alla definizione di un algoritmo "ottimizzatore", che costituisce l'obiettivo del presente capitolo, è necessario percorrere un certo iter concettuale, che prende le mosse da un modello di RNA che, più che all'ottimizzazione, è dedicato al riconoscimento di esempi. Esso definisce, infatti, la struttura topologica da impiegare nel modello definitivo che si otterrà. A cambiare sostanzialmente, passando dal prototipo al risultato finale, saranno alcuni accorgimenti che intervengono nell'aggiornamento delle attivazioni neurali. Una rete adibita all'ottimizzazione, infatti, deve essere addestrata a riconoscere gli esempi sulla base di "compatibilità", più che di "somiglianze". Per questo motivo, essa, anziché determinare la risposta confrontando l'input ricevuto con l'insieme degli esemplari che "conosce", la determina "valutan-

do" quanto bene lo stesso input si adatti alle condizioni da soddisfare: "analisi" più che "riconoscimento".

In realtà, comunque, non è l'intera classe delle RNA applicate al pattern recognition a servire da modello. È sufficiente, piuttosto, osservarne una sottoclasse, quella delle **memorie autoassociative**. Non sarà quindi necessario addentrarsi troppo a fondo nel campo del riconoscimento, ma semplicemente dedicare un po' di attenzione ai criteri di funzionamento ed ai principi che stanno alla base della summenzionata tipologia specifica di sistemi neurali.

4.1 Memorie autoassociative

Il pattern recognition, come si è detto più volte, ha il suo fondamento nella presenza di un insieme fissato di immagini che fungono da esemplari, "noti" alla rete neurale. Ogni volta che, nel modo opportuno, viene proposta un'immagine *non compresa* nell'insieme degli esemplari, la rete cerca tra questi ultimi quello che a tale immagine le sembra più "simile". Il termine "immagine", è il caso di ribadirlo, non indica qui un'impressione visiva, ma piuttosto *qualsiasi insieme di segnali, provenienti da una sorgente, che ne rappresentano caratteristiche specifiche utili al suo riconoscimento.* Il concetto di "somiglianza", in questi termini, è relativo soltanto alle caratteristiche che vengono, volta per volta, valutate. Per fare un esempio concreto, se una rete neurale dovesse classificare gli strumenti musicali in base al loro timbro, potrebbe ritenere "simili" due strumenti come violoncello e contrabbasso, nonostante la loro differenza nelle dimensioni e nell'estensione armonica, dato che, frequenza a parte, la forma dell'onda sonora prodotta da entrambi è quasi la stessa.

Chiaramente, con questo tipo di performance, le reti neurali artificiali intendono riprodurre il processo secondo cui il cervello, analizzando le percezioni ricevute dall'esterno, le confronta con quelle analoghe registrate nella propria memoria, riconoscendo così la sorgente dei segnali ricevuti. In certi casi, e qui entra in gioco l'ottimizzazione, il cervello non riesce, con la sola analisi mnemonica, a ricostruire l'oggetto, ed allora opera una serie di "adattamenti" al segnale captato, in modo da tentare di "completare" l'immagine. Ritornando all'esempio del suono, si potrebbe innescare questo processo quando si volesse tentare di riconoscere il timbro di uno strumento musicale, ma dopo averlo ascoltato da un supporto poco fedele.

Il modello di rete neurale di cui si intende discutere adesso, è un'applicazione delle operazioni di riconoscimento descritte qui in modo qualitativo. Prima di discutere la struttura di questa particolare RNA, allora, è utile proporre un semplice schema matematico, che consenta di esprimere lo stesso discorso proposto in questo paragrafo con concetti più "quantitativi", che permettano di tradurre in termini matematicamente più definiti i concetti di "immagine" e "somiglianza", per poterli poi applicare con maggior cognizione di causa ai dispositivi fisici di cui si dovrà discutere.

4.1.1 Spazio di Hamming

Lo **spazio di Hamming** è un sottoinsieme dello spazio $\mathbf{R}^n$ delle n-uple di numeri reali, che viene definito come segue [Free91]:

$$\mathcal{H}^n = \{\mathbf{x} \equiv (x_1, x_2, \ldots, x_n) \in \mathbf{R}^n : x_i \in \{0,1\} \forall i = 1, \ldots, n\} \tag{4.1}$$

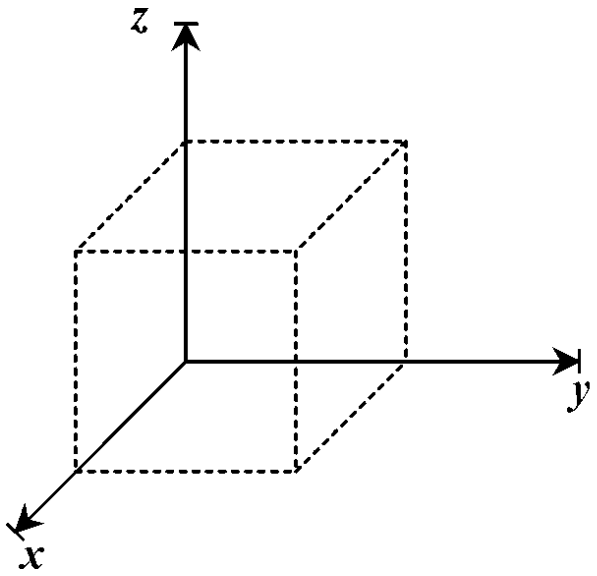

Figura 4.1: Il cubo di Hamming nello spazio a 3 dimensioni. Gli spigoli del solido, rappresentato con i tratteggi, sono i vettori dello spazio in questione.

Si tratta, quindi di tutte le n-uple le cui componenti valgono 1 o 0[1]. Nel piano cartesiano a dimensione n, esse sono gli spigoli dell'ipercubo che ha uno spigolo nell'origine e le facce parallele ai piani coordinati:

La distanza euclidea tra due spigoli $\mathbf{x}$ e $\mathbf{y}$ vale :

$$d = \sqrt{\sum_i (x_i - y_i)^2} \tag{4.2}$$

Dati i valori caratteristici degli elementi dello spazio di Hamming, tuttavia, è evidente che ogni addendo vale 0 quando due componenti omonime sono uguali, ed 1 nel caso contrario. Tale osservazione pone la distanza (4.2) in relazione con il numero delle coppie di componenti che hanno uguale indice e diverso valore. Nota la quantità h di tali coppie, infatti, è immediato che:

$$d = \sqrt{h}$$

La quantità h viene chiamata distanza di Hamming tra i due vettori.

Si immagini adesso, nello spazio di Hamming ad n dimensioni, di individuare alcuni elementi, indicati con $\mathbf{x}_j$, per $j = 1, \ldots, N$, che verranno denominati *esemplari*. Fatto ciò, si costruisca la funzione $F : \mathcal{H}^n \to \mathcal{H}^n$, definita come segue:

- se l'argomento $\mathbf{x}$ è un esemplare $\mathbf{x}_j$, F restituisce quest'ultimo;
- se l'argomento $\mathbf{x}$ non è un esemplare, F restituisce l'esemplare che, in termini di distanza di Hamming, è il più vicino a $\mathbf{x}$.

[1]Come è possibile confrontare nel riferimento citato, in realtà lo spazio di Hamming trattato nella letteratura conterrebbe i vettori con coordinate uguali a 1 e -1, anziché 1 e 0. Tuttavia, nell'intento di rendere più adatto il discorso a quelle che saranno le applicazioni pratiche di tale teoria in questa tesi, risulta più conveniente usare la trattazione presentata nel testo.

Con questo procedimento, in pratica, si suddivide l'intero spazio di Hamming in N classi, ciascuna delle quali contiene tutti gli elementi dell'insieme che sono vicini ad un particolare esemplare più che a qualunque altro. Una funzione che operi questo tipo di suddivisione è chiamata **autoassociazione lineare** dello spazio di Hamming su se stesso.

Nell'ottica di questa discussione matematica, è chiaro come sia possibile spiegare i processi concettuali esposti, a titolo d'introduzione, in precedenza. Tutte le "immagini" sono vettori di Hamming e si mostrerà fra breve in che modo sono implementabili in una rete neurale artificiale. Tra essi, gli esemplari dell'autoassociazione costituiranno naturalmente le "conoscenze" della rete, a cui ricondurre ogni altra immagine secondo i canoni delle funzioni autoassociative discusse sopra. Infine, il concetto di "somiglianza" può essere rappresentato debitamente dalla distanza di Hamming tra due vettori, visto il ruolo che essa ricopre nella determinazione della classe in cui collocare ogni vettore "non esemplare".

4.1.2 Struttura delle memorie autoassociative

Voler rappresentare il fatto che una rete neurale esegua un'autoassociazione nello spazio di Hamming, permette di individuare innanzitutto le caratteristiche essenziali della struttura topologica e dell'hardware di questi sistemi.

La prima cosa, estremamente evidente, è che i vettori di Hamming sono binari. Quindi, visto che le componenti sono rappresentate dai neuroni, si useranno neuroni binari, quelli a risposta 0/1, o comunque, riconducibili in ultima analisi a questo schema[2]. Questa necessità comporta, tra l'altro, la necessità di tenere presente una soglia d'attivazione per ciascuna delle unità neurali.

In secondo luogo, è evidente che, in questa situazione, input ed output appartengono allo stesso insieme statistico, per cui l'input layer e l'output layer debbono essere collegati da una precisa corrispondenza uno ad uno. Tale corrispondenza comporta anzitutto la necessità che i due strati abbiano lo stesso numero di unità. Inoltre, bisogna creare una correlazione biunivoca tra strato di input e strato di output, che metta in relazione una coppia di unità, prendendone una da ogni strato, con ogni specifica componente di un generico vettore dello spazio di Hamming di dimensione pari al numero N dei neuroni.

Per quanto esistano, come si vede, alcuni princìpi da rispettare, una RNA autoassociativa è realizzabile con entrambe le modalità topologiche proposte nel capitolo 2. Quando si usa la topologia a strati, le identità e le corrispondenze tra input ed output layer dovranno essere stabilite *ad hoc*. La presenza di strati nascosti, in questo caso, consente le migliorie tipicamente riconducibili ad essi, ossia la maggior specificità nel riconoscimento delle differenze tra esempi poco diversi. Questo rende tali reti neurali molto utili nel caso in cui gli esempi siano definiti a priori, ossia quando lo scopo sia più specificamente quello di operare un pattern recognition.

Si è detto che questa, tuttavia, è una discussione in cui tale procedimento serve solo da punto di partenza per giungere a situazioni in cui gli esemplari non sono definiti, perché non esistono in realtà. Per questo motivo, non è il caso di soffermarsi sul modello a strati, che tra l'altro comporta difficoltà nell'implementazione della sua simulazione al computer. L'obiettivo della prossima parte sarà, invece, il modello a strato singolo di rete neurale

[2]Si potrebbero — e lo si farà in seguito — usare anche neuroni ad attivazione reale e limitata tra 0 ed 1, e poi stabilire un valore al di sopra del quale ritenere il neurone "attivo", per cui convertire le attivazioni finali in 0 od 1 secondo il confronto dello stato finale della rete con tale parametro, posto in genere a 0.5, che è il valore intermedio.

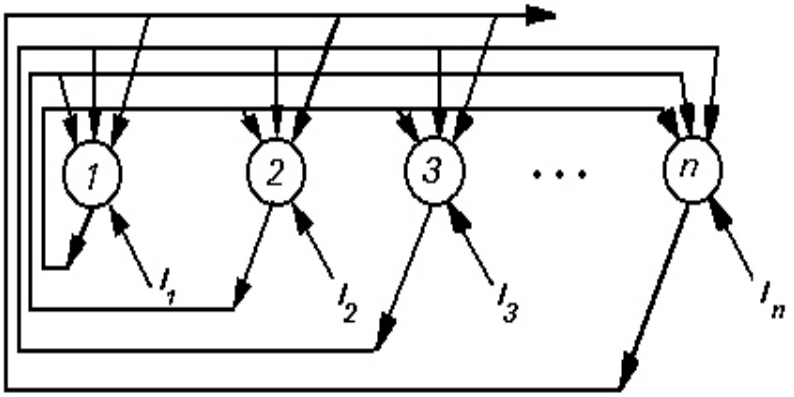

Figura 4.2: Schema topologico di una rete di Hopfield, che evidenzia la presenza di connessioni in ingresso ed in uscita di ogni unità con ogni altra. Le I_n rappresentano dei segnali che fungono da soglia di attivazione, implementati con neuroni che inviano sempre un segnale uguale ad 1.

autoassociativa, che è noto come **rete di Hopfield** dal nome del biologo che ne propose per primo il modello [Kot93, Bol95, Free91].

4.2 La rete di Hopfield

4.2.1 Modello discreto

Una rete neurale di Hopfield è un sistema a strato singolo completamente connesso, con unità ad attivazione binaria a soglia. Le connessioni sono bidirezionali per ogni coppia di neuroni, e non è impossibile trovare connessioni di feedback tra ogni unità e se stessa. In genere, la matrice dei pesi sinaptici, oltre che quadrata, come è ovvio che sia, viene resa simmetrica, in modo da rendere ogni peso indipendente dal senso in cui viene percorsa la connessione cui è associato. Per distinguere questo modello da altri che si incontreranno in seguito, che sono caratterizzati da neuroni la cui attivazione è un numero reale, sarà il caso di indicare, più esattamente, questo modello col nome di *rete di Hopfield discreta*. Anzi, a questo proposito, è il caso di anticipare che l'ottenimento di una RNA con attivazioni reali è proprio lo scopo finale del discorso che in questo capitolo si intende proporre, ma solo dopo aver fatto diverse considerazioni necessarie per dare senso a questa scelta.

La figura 4.2 rappresenta lo schema di una generica rete di Hopfield.

In generale, chiamando net_i l'input complessivo ricevuto dall'i-esimo neurone e $\mathbf{w}$ la matrice dei pesi sinaptici, vale la seguente espressione:

$$net_i = \sum_k w_{ik} a_k + I_i \tag{4.3}$$

ossia la somma pesata delle attivazioni degli altri neuroni, più un termine di input proveniente da un neurone che è sempre attivo, e funge da soglia. Per questo motivo, si può proporre un'alternativa a tale ragionamento in cui tale input esterno sia posto uguale a zero e si consideri una soglia S_i diversa da zero per ogni neurone, senza per questo ledere la generalità del discorso.

L'aggiornamento dell'attivazione dell'i-esimo generico neurone, secondo quanto è previsto dal funzionamento descritto al capitolo 2 circa i neuroni a soglia, prevede il calcolo

di net_i ed il suo successivo confronto con S_i, da cui discende l'attivazione — che in questo caso è uguale all'output, per cui la si indica con la lettera x riservata appunto all'output nei casi mostrati in precedenza:

$$x_t(t+1) = \begin{cases} 1 & \text{se} \quad net_i > S_i \\ 0 & \text{altrimenti} \end{cases} \tag{4.4}$$

Come sempre, si pone un incremento temporale discreto in cui il tempo indica il ciclo progressivo d'aggiornamento.

La definizione dei pesi sinaptici è un processo determinato dal criterio che si vuole usare. Una rete di Hopfield, infatti, può essere impiegata tanto con un addestramento quanto senza. Quest'ultima situazione sarà trattata più avanti; qui basti dire che l'addestramento viene sostituito, in quel caso, da criteri logici che consentono di definire i pesi a priori, secondo considerazioni fatte a partire dal problema che si intende risolvere.

Nel caso in cui la rete di Hopfield debba essere addestrata con esemplari definiti, il metodo più comune con cui essi vengono "insegnati" alla rete, è la regola di Hebb [Hum90]. Questo metodo di aggiornamento, come si è definito nel capitolo 2, prevede di proporre alla rete i diversi esemplari in input, aggiustando i pesi direttamente — ossia *senza che la rete si aggiorni* — variando ciascuno di una quantità proporzionale al prodotto delle attivazioni dei neuroni che collega. Definendo un parametro di training che controlla l'entità delle variazioni, chiamato in genere η, l'espressione è:

$$\Delta w_{ij} = \eta \left(x_i - \frac{1}{2}\right)\left(x_j - \frac{1}{2}\right) \tag{4.5}$$

in cui si è leggermente diversificata l'espressione rispetto a quella analoga riportata nel capitolo 2, in quanto qui la correzione deve essere positiva e diversa da zero anche quando i due neuroni si trovano ad attivazione 0 entrambi. Inoltre, la correzione deve essere negativa quando un'unità si trova nello stato 1 e l'altra nello stato 0. Se si usasse semplicemente il prodotto delle attivazioni, ogni volta che si dovesse considerare un elemento ad attivazione zero, esso annullerebbe la correzione, mentre con la (4.5) si raggiungono entrambi i risultati discussi adesso.

Si può anche fare un compromesso tra l'addestramento con esempi definiti e la regolazione a priori dei pesi [Free91], che consiste, dato un determinato campione di vettori d'attivazione, nel creare la matrice di un'applicazione lineare che riconduca ciascuno di questi vettori in se stesso. Nel caso in cui si possa esprimere l'insieme degli esemplari con una base ortonormale dello spazio di Hamming rappresentato dalla rete stessa, si dimostra che tale matrice è:

$$\mathbf{w} = \sum_k \mathbf{x}_k \mathbf{x}_k^t$$

ove con k si indica l'indice progressivo dell'esemplare.

Il processo che avviene per l'aggiornamento della rete è semplice: innanzitutto viene fornito un vettore d'attivazione, ossia un vettore $\mathbf{x}$ dello spazio di Hamming che rappresenta lo stato iniziale d'attivazione degli N neuroni. Fatto questo, inizia il processo di autoaggiornamento, in cui ogni neurone determina il proprio input complessivo con l'equazione (4.3) e poi determina la sua nuova attivazione con la regola (4.4). Naturalmente, in base a questi due passaggi, l'attivazione può cambiare, come può anche non farlo: è questa la base dell'utilità di tutto lo schema che si sta presentando. Il meccanismo di aggiornamento, infatti, prevede proprio che, guidate dai valori dei pesi, le attivazioni tendano

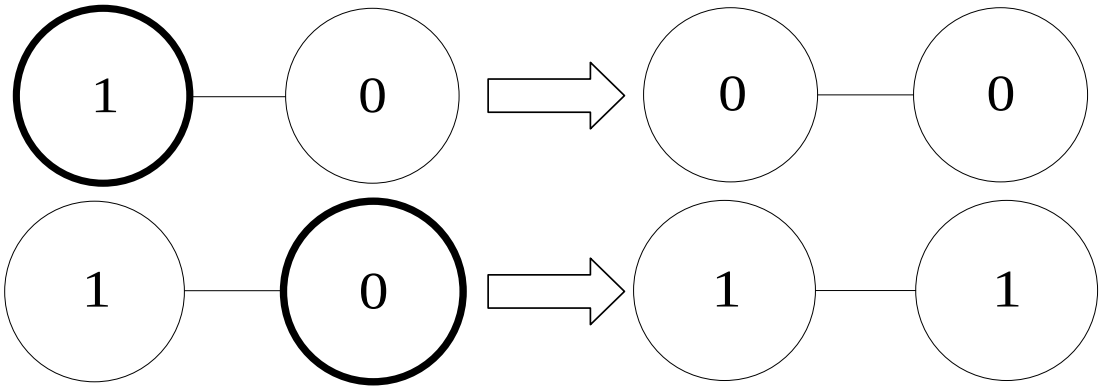

Figura 4.3: Aggiornamento sequenziale nell'esempio proposto nel testo a partire da ciascuna delle due unit'a.

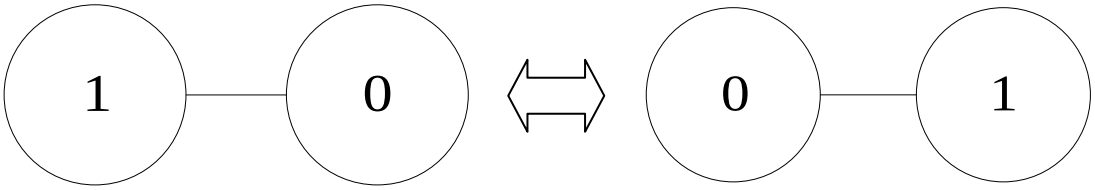

Figura 4.4: Lo stesso caso precedente, ma con aggiornamento parallelo.

ad assestarsi verso un esemplare (se è definito) o comunque verso una configurazione tale che, di ciclo in ciclo, il numero di attivazioni modificate decresca continuamente, fino ad annullarsi. Giungendo a questo punto, un ulteriore ciclo di aggiornamento non produce più alcuna modifica al vettore d'attivazione ottenuto: è stato raggiunto quello che si suole definire uno *stato stabile*, che è l'output definitivo della rete di Hopfield.

Il criterio con cui avviene l'aggiornamento della rete può essere *seriale* o *parallelo*. Il primo, come il nome lascia intendere, consiste nell'aggiornare un elemento per volta, sicché ognuno viene influenzato dalla situazione che trova nella rete in quel momento. In questo caso, dopo ogni singolo calcolo, non sempre tutti i neuroni sono stati aggiornati lo stesso numero di volte, ma ciò non influenza molto il risultato. Esistono comunque casi in cui la sequenza di aggiornamento potrebbe determinare la risposta. Un esempio molto banale è il caso di una rete costituita da due neuroni con la stessa soglia S ed un peso w di connessione maggiore di S. Se si pone uno dei due a 0 e l'altro ad 1, aggiornando per primo quello a 0 succede che esso riceve un input uguale a w, che, essendo maggiore di S, lo "accende". Dopo, aggiornando l'altro, questo riceverà lo stesso input, e quindi rimarrà acceso anch'esso. Si può vedere che questo stato è stabile, per cui partendo con l'aggiornamento del neurone spento, si arriva allo stato $(1, 1)$. Se adesso, invece, si inizia con il neurone posto a 1, esso riceve un input nullo, essendo l'altro a 0, per cui si spegne, e ciò, è evidente, comporta il successivo mantenimento a 0 dell'altra attivazione. La figura 4.3 ne da una rappresentazione grafica schematica, in cui con la linea spessa si indica il neurone aggiornato per primo.

Si vede nella figura come, dalla stessa configurazione, è possibile ottenere due diversi stati finali, in seguito al diverso ordine di aggiornamento. Naturalmente, tali forti dipendenze vengono molto indebolite quando i neuroni sono tanti, ma potrebbe essere utile evitare di aggiornare i neuroni sempre nello stesso ordine, almeno quando si tratta di neuroni binari, come in questo caso. Nonostante tale punto da tenere in considerazione, questo criterio di aggiornamento è molto spesso il migliore, in quanto alcune considerazioni che si vedranno più avanti, evidenziate dallo stesso Hopfield (cfr. appendice D), mostrano come

esso conduca sempre ad uno stato stabile.

Il criterio parallelo, a differenza del precedente, aggiorna tutti i neuroni assieme, nel senso che, ad ogni ciclo, le nuove attivazioni divengono effettive solo dopo essere state calcolate per tutti i neuroni. Uno studio dovuto a Goles et al. [Kot93] dimostra che anche questo modello può condurre ad uno stato stabile, ma non sempre. Infatti, stavolta, c'è la possibilità di raggiungere un'oscillazione tra due stati, ciascuno dei quali si ottiene dall'altro dopo un aggiornamento, e quindi non si ottiene mai la stabilizzazione. Un esempio estremamente semplice si può fare con la stessa rete neurale a due elementi dell'esempio precedente. Se si parte dalla configurazione $(1, 0)$, con questo criterio di aggiornamento il primo neurone riceverà un input uguale a 0 e quindi anch'esso si porterà a 0, mentre l'altro riceverà un input uguale a w, che, se w è maggiore della soglia S, conduce all'attivazione 1. Al ciclo successivo, ripetendo il ragionamento, si torna alla configurazione di partenza, e così via all'infinito. Si noti come, in ogni caso, gli stati ottenuti non corrispondano a nessuno di quelli che il criterio seriale restituisce. Nella figura 4.4 si trova un'illustrazione visuale di ciò che succede.

4.2.2 La funzione energia

Questo paragrafo è dedicato a spiegare un concetto importante, in base al quale Hopfield poté concludere che il suo modello di rete neurale, se impostato con pesi simmetrici ed aggiornato sequenzialmente, conduce sicuramente ad uno stato stabile. Da esso, poi, discende la definizione di una funzione dipendente dallo stato di attivazione della rete neurale. Tale funzione è utile in quanto ricopre un ruolo importante nell'iter che conduce alla determinazione del modello di rete neurale che costituisce l'obiettivo della presente capitolo.

Il concetto in questione è l'appartenenza della rete di Hopfield discreta, simmetrica e con aggiornamento sequenziale, alla classe di sistemi che soddisfano il *criterio di Lyapunov* [Free91]. Questo criterio qualitativo dipende dalle variabili che determinano la dinamica dello stato di un sistema, quindi, in questo caso, le attivazioni neuroniche. Esso assicura che, se esiste una funzione di queste variabili, la quale:

- sia limitata per ogni configurazione del sistema;
- decresca ad ogni cambiamento di configurazione;

allora il sistema in esame, dopo un certo numero di aggiornamenti del proprio stato, convergerà certamente verso una stabilizzazione, in cui la configurazione raggiunta non sarà più soggetta a variazioni.

Nel caso delle reti neurali, infatti, è possibile definire questa funzione, che, per motivi da chiarire più avanti, anche se la sua forma ne mostra già alcuni (i termini di energia cinetica nelle equazioni del moto in meccanica) viene solitamente chiamata **energia**. La si esprime come segue:

$$E = -\frac{1}{2}\mathbf{xwx} = -\frac{1}{2}\sum_{i,k} w_{ik}x_i x_k \tag{4.6}$$

L'energia è, quindi, una forma quadratica nelle attivazioni. È possibile mostrare (appendice D) che, nel caso di una rete di Hopfield ad aggiornamento seriale e con pesi simmetrici, infatti, l'energia della rete decresce per ogni ciclo di revisione delle attivazioni. Il fatto che E sia limitata discende dalla considerazione che i pesi sono valori fissati e le

x_i possono valere 0 od 1, per cui quest'energia, può avere valori compresi tra la somma di tutti i pesi sinaptici positivi e quella di tutti gli analoghi negativi.

Il fatto che l'energia decresca mentre la rete si sposta verso uno stato stabile, implica che le configurazioni finali debbano corrispondere ai minimi relativi di questa funzione: è l'unico modo di conciliare matematicamente il principio di Lyapunov con quanto detto finora. Per questo motivo si potrebbe rappresentare l'aggiornamento, in un grafico dell'energia in funzione dello stato d'attivazione del sistema, come un processo di "attrazione" graduale verso il fondo di una buca di potenziale. In quest'ottica, quando la rete raggiunge una configurazione posta ben dentro una di queste buche, vi rimane intrappolata e tende quindi a portarsi verso il suo minimo relativo, impossibilitata ad uscirne dalla necessità di far decrescere l'energia ad ogni passaggio successivo. Non importa se i diversi minimi sono posti a valori diversi: "comanderà" sempre quello della prima buca che intrappola la rete. Naturalmente, ogni buca sarà tanto più favorita quanto più si trova vicina allo stato iniziale della rete, per cui, come è previsto e normale in questo tipo di sistema, la risposta dipende essenzialmente dall'input. Altrettanto ovvio è che la "risoluzione" con cui la rete sa distinguere due input poco differenti dipende certamente dal numero di minimi nell'energia, ma anche dalla loro distanza: due minimi troppo vicini e separati da un massimo relativo troppo basso rendono difficile alla rete "intrappolare" lo stato del sistema, e la rete rimane a lungo "indecisa", il che potrebbe determinare un certo margine d'errore nella risposta.

Un ultimo vantaggio che l'energia permette di acquisire sarà espresso meglio all'ultimo stadio dell'iter che condurrà all'algoritmo definitivo. Per adesso basti dire che, mantenendo limitata la funzione energia, si possono aggiungere ad essa dei termini che comportino la soddisfazione di eventuali vincoli sulla soluzione. È questo il modo di implementare l'ottimizzazione a priori.

4.2.3 Variante continua del modello di Hopfield

Una variante della rete neurale discussa finora prevede neuroni la cui funzione d'attivazione sia non più binaria, ma sigmoidale, e per questo motivo si parla di *rete di Hopfield continua*. Uno dei vantaggi maggiori di questo modello è il fatto che rappresenta più da vicino il segnale che è possibile ottenere in pratica con dispositivi elettronici. Inoltre, il suo funzionamento consente un maggiore controllo sulle variazioni dell'energia e sul processo di aggiornamento, entrambe cose che derivano dal fatto che si passa da valori discreti a valori continui.

Se u è l'input ricevuto dal neurone, in questo nuovo modello la sua attivazione sarà data dalla funzione nota come "logistica":

$$a(u) = \frac{1}{1 + e^{-\frac{u}{T}}} \tag{4.7}$$

ove $1/T$ è chiamato *parametro di guadagno*.

Questa funzione è limitata tra 0 ed 1, e somiglia molto ad un gradino, specie per coefficienti di guadagno grandi. Essa rappresenta, anzi, un gradino vero e proprio nel limite in cui $T \to 0$.

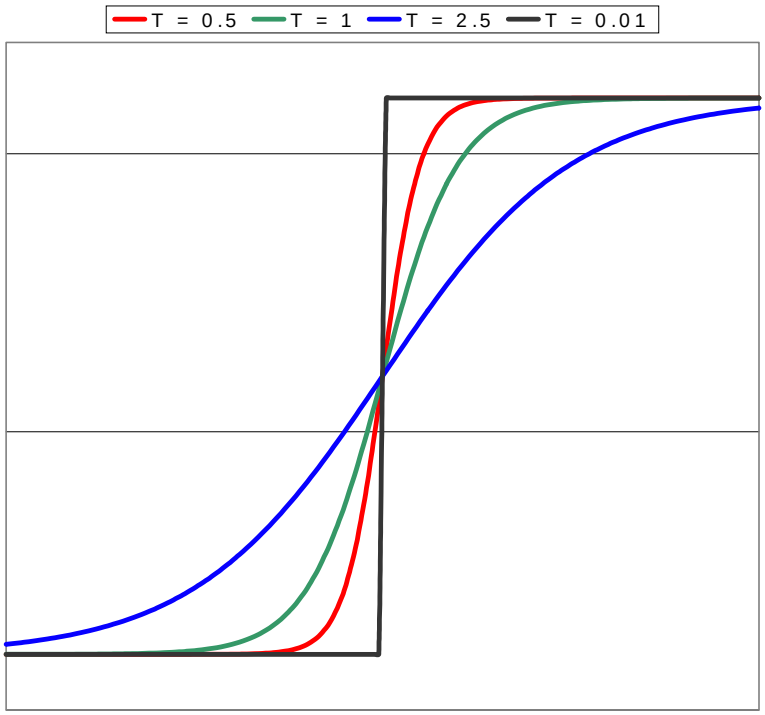

Figura 4.5: Andamento della funzione logistica per diversi valori di T. Si nota come, per valori piccoli, essa tenda al gradino.

4.3 Problemi di ottimizzazione

Una volta introdotto lo schema topologico da usare e la funzione energia soprattutto, si passa al problema fondamentale che sorge in questo tipo di rete. Si è detto che una rete neurale di Hopfield converge verso un minimo dell'energia, ma che tale minimo non è sempre quello assoluto. Essa cerca semplicemente un punto stazionario, che in genere viene scelto tra i più vicini alla configurazione iniziale. Ora, questo comportamento può andar bene nel caso di un pattern recognition, in cui i "pattern", per l'appunto, sono tanti. Anzi, in questo caso, si è dedotto come a ciascun esemplare corrisponda un minimo relativo dell'energia e quindi, in questo caso, è giusto ed auspicabile che la rete restituisca il minimo più "vicino", in termini energetici, all'esempio iniziale proposto.

Nel campo dell'ottimizzazione, invece, ciò che si richiede alla rete è di fornire solo e sempre *la soluzione migliore* al problema posto. Un esempio, citato spesso in trattati relativi alle reti neurali artificiali [Free91, Kir83], è il "problema del commesso viaggiatore" (abbreviato in TSP, dalla traduzione inglese "Traveling Salesman Problem"). Questo problema consiste nello stabilire, date N città e note le distanze tra ciascuna coppia di esse, in che ordine dovrebbe visitarle un commesso viaggiatore, se egli volesse andare in ciascuna una volta sola e tornare al punto di partenza, il tutto compiendo il percorso più breve. È chiaro come, in questo caso, la rete debba dare una ben precisa risposta: *il percorso più breve*, e si presuppone che non ne esistano due diversi egualmente buoni, a parte il fattore di degenerazione di ordine N dipendente dalla scelta della città di partenza ed a quello di ordine 2 dovuto alla scelta del senso in cui il percorso viene seguito. Questo è un esempio di **problema NP-completo** [Kir83], ossia un problema che, se dipende da M variabili, ha una soluzione che richiede un numero di calcoli che cresce con M più rapidamente di qualunque polinomio di ordine M. Una delle caratteristiche di questa tipologia di proble-

mi è quella di divenire estremamente "faticosi" per qualsiasi computer quando M diventa molto grande. A questa classe, tra l'altro, appartiene anche il problema del riconoscimento di tracce nei rivelatori, cui si porrà l'attenzione nel prossimo capitolo.

L'unico punto dell'energia della rete neurale che risulta, per qualche verso, "unico" in termini del significato che questa funzione assume nei confronti del sistema, è il suo minimo assoluto. Quindi, in termini di questa funzione si può tradurre il problema dell'ottimizzazione in quello della ricerca della minimo assoluto della funzione energia. Naturalmente, i pesi sinaptici dovranno essere definiti opportunamente, in modo che tale minimo corrisponda effettivamente alla soluzione cercata.

Si è detto, tuttavia, che la rete discreta di Hopfield, a causa del suo principio di aggiornamento, crea intorno ad ogni suo minimo energetico una buca d'attrazione, di modo che, una volta entrata in qualunque di esse, lo scopo del sistema è quello di raggiungerne il fondo, a prescindere dal fatto che sia assoluto o relativo. È chiaro, quindi, che la prima cosa da fare è superare il processo d'aggiornamento di Hopfield, introducendone uno che consenta alla rete di "saltare" da una buca all'altra, fino a trovare quella più profonda, e solo dopo stabilizzarsi sul suo fondo.

Il metodo più accreditato per conseguire questo risultato è ispirato ad un procedimento che ha un analogo fisico in un metodo di lavorazione dei materiali, usato per produrre cristalli strutturalmente il più perfetti possibile: l'**annealing**. Per implementare in una rete neurale un simile concetto, però, bisogna fare in modo da creare un parallelo tra meccanica statistica e questo campo della teoria dell'informazione. A questo scopo, si procederà ad un breve richiamo di alcuni concetti di meccanica statistica classica che risultano utili allo scopo. Fatto questo, sarà possibile proporre un criterio di aggiornamento che possa essere considerato un "annealing simulato", per ottenere un'evoluzione della rete di Hopfield che sia adatta a risolvere problemi di ottimizzazione.

4.3.1 Richiami di meccanica statistica classica

La classificazione consueta che la Meccanica Statistica opera, distingue tre classi di insiemi statistici, in base alle grandezze caratteristiche che ne definiscono lo stato macroscopico: l'insieme *microcanonico* è un sistema isolato che non scambia con l'esterno né energia, né particelle; l'insieme *canonico* può scambiare solo energia, e l'insieme *grancanonico* può scambiare sia energia che particelle. Ne consegue [Hua97] che le grandezze costanti che caratterizzano uno stato macroscopico sono differenti, secondo il tipo di statistica d'insieme applicata. Nel caso di una rete neurale, l'energia varia, anzi deve farlo, in base a quanto visto, ma il numero di elementi deve, per forza di cose, mantenersi costante. Queste considerazioni bastano a spiegare perché sia il caso di rivolgere l'attenzione alla statistica dell'insieme canonico.

L'insieme canonico, proprio a causa del fatto che può scambiare solo energia con l'esterno, non può essere caratterizzato da un valore costante di essa. Invece, può essere un buon parametro caratterizzante la temperatura, che, mantenendosi costante, può tuttavia permettere di pensare il nostro sistema come un oggetto che scambia energia con un serbatoio di calore, per mantenersi in equilibrio termico con esso [Pat87].

Il fatto che l'energia di un insieme canonico sia variabile, pone il problema di conoscere quale sia la probabilità di trovare, in un qualsiasi istante t, il sistema in esame in uno degli stati caratterizzati da un preciso valore dell'energia totale. Per farlo, si può procedere in due modi:

- studiare la statistica dello scambio di energia con il serbatoio di calore a temperatura costante;
- studiare la statistica generica di questo tipo d'insieme sostituendo poi i valori opportuni come parametri della distribuzione di probabilità conclusiva.

Considerando il primo caso, bisogna immaginare definita la temperatura T di equilibrio, e l'energia complessiva dei due sistemi, E_{tot}, che può essere variabilmente ripartita tra i due sottosistemi considerati (il serbatoio di calore e il sistema in esame). L'insieme dei due sistemi parziali è un sistema isolato, la cui energia è costante, per cui la ripartizione delle energie è comunque vincolata a mantenere costante la somma delle due, ma il fatto che uno dei due sia un serbatoio di calore, e quindi sia in grado di non risentire in modo sensibile anche di scambi di energia notevole, si può pensare che l'energia del sistema in esame sia molto minore di quella del serbatoio. Quest'ultimo si può trovare in uno qualsiasi degli stati compatibili con la "propria" frazione di energia interna, tutti egualmente probabili, per cui la probabilità che esso abbia un certo valore di tale energia interna è direttamente proporzionale al numero di questi stati. Da una serie di calcoli [Pat87, Hua97], si riesce a determinare la seguente distribuzione di probabilità, che dipende dall'energia E del sistema e non da quella del serbatoio:

$$P(E) = \frac{e^{-\frac{E}{kT}}}{Z} \tag{4.8}$$

Z è una somma su tutti i possibili stati alle diverse energie, che si chiama **funzione di partizione canonica** e serve a normalizzare la probabilità. La forma caratteristica della distribuzione, poi, è fondamentale per calcolare il valor medio di ogni grandezza fisica associata a questo insieme. Ad esempio, l'energia media vale:

$$U = \langle E \rangle = \frac{\sum_E E e^{-\frac{E}{kT}}}{Z} = -\frac{\partial}{\partial(1/kT)} Z$$

Facendo adesso alcune considerazioni per dedurre le caratteristiche fisiche del sistema in esame, è possibile trarre i princìpi da applicare alle reti neurali. In tutto il discorso la costante di Boltzmann k verrà considerata uguale ad 1. Il primo passo è quello di definire l'energia libera del sistema, A:

$$A = -T \ln \left[\sum_E e^{-\frac{E}{T}} \right] \tag{4.9}$$

che si dimostra [Pat87] essere uguale a $U - TS$, ove S è l'entropia del sistema. Questa conclusione si deduce direttamente, ricordando la definizione statistica consueta dell'entropia, correlata con la probabilità:

$$S = -k \sum_E P(E) \ln P(E)$$

Sostituendo, infatti, la forma di P trovata e considerando la Z come una costante, si ottiene immediatamente che:

$$S = -\sum_E \frac{e^{-\frac{E}{T}}}{Z} \left(-\frac{E}{T} - \ln Z \right) = \frac{U - A}{T} \tag{4.10}$$

che è la stessa relazione $A = U - TS$, risolta rispetto a S. In questo modo si evidenzia la fondamentale importanza della funzione di partizione nella determinazione delle caratteristiche salienti del sistema (dette funzioni di stato), a temperatura definita. Nel caso

di interesse in questa sede, tali considerazioni vanno applicate ad un insieme canonico che funge da base per un modello, proposto da Ising per spiegare il comportamento dei domini ferromagnetici [Hua97]. Questo modello descrive un materiale ferromagnetico come un reticolo di elementi la cui unica variabile di stato da considerare è lo spin, il quale, a sua volta può avere solo due valori: su (1) o giù (–1). L'energia di questo sistema viene descritta dalla seguente formula:

$$E = \sum_{i,j} E_{ij} s_i s_j - H \sum_i s_i \tag{4.11}$$

in cui H è il campo magnetico esterno ed E_{ij} è l'energia di interazione tra due siti reticolari collegati. Se si tratta il problema con la statistica classica di Boltzmann, vuol dire che è possibile distinguere due siti differenti, per cui uno stato è assegnato se si conosce il vettore $\mathbf{s}$ in cui di *ogni sito* è stabilito lo spin. In quest'ottica, lo stato fondamentale è degenere di un fattore 2 per ogni coppia di spin che è possibile trovare, in questo stato, nelle condizioni di spin opposti ed interazione non nulla. Il motivo è che scambiandone gli spin, evidentemente, si ottiene uno stato che in linea di principio è diverso e distinguibile dal primo, ma possiede la stessa energia.

La funzione di partizione di un sistema del genere, è la seguente:

$$Z = \sum_{\mathbf{s}} e^{\frac{E(\mathbf{s}}{T}} \tag{4.12}$$

ove si è specificato nell'indice di sommatoria un indice vettoriale che scorre tutti i 2^N possibili modi di disporre gli spin, dove N è il numero di siti reticolari presenti. Da Z discende A e poi le altre funzioni importanti del sistema come descritto precedentemente. Ciò che è importante è la notevole rassomiglianza tra l'energia di un sistema di Ising e quella della rete di Hopfield, fatte le debite considerazioni, cui è dedicato il prossimo paragrafo.

4.3.2 Annealing simulato

Il modello di Ising si può ricondurre immediatamente al caso di una rete neurale di Hopfield. Basta vedere le attivazioni neurali come spin, i pesi delle connessioni come energie d'interazione, e notare come corrisponda l'energia di Hopfield con quella di un dominio sottoposto ad un campo magnetico H nullo.

Il parallelismo è quindi chiaro: nel caso di una rete neurale di Hopfield, l'insieme statistico è dato dallo spazio di Hamming dei possibili vettori di attivazione della rete, esemplari o meno. La base teorica di un modello statistico d'insieme prevede che un sistema possa trovarsi in uno qualunque degli stati microscopici compatibili con la sua temperatura e con le altre variabili macroscopiche definite, per cui lo stato microscopico è soggetto a fluttuazioni statistiche. In quest'ottica, allora, è necessario pensare di introdurre, nell'aggiornamento della rete, indeterminazioni dovute a queste fluttuazioni. In altre parole, bisogna che il criterio di aggiornamento, se vuole introdurre i concetti relativi alle indeterminazioni tipiche di questi sistemi, aggiunga fasi in cui l'aggiornamento dipenda da estrazioni casuali, pesate sulle distribuzioni statistiche d'insieme. Ciò si realizza mediante l'adozione di una scelta casuale, al posto di quella deterministica che governa l'evoluzione della rete esposta in precedenza.

Come si è introdotto prima, esiste un corrispettivo fisico del modello informatico da utilizzare, ossia il processo di annealing. Esso viene applicato nelle situazioni in cui c'è la necessità di produrre cristalli il cui reticolo microscopico sia il più regolare possibile. Se,

infatti, si produce un cristallo raffreddando bruscamente il suo liquido, si rischia di fissare le posizioni reticolari dei suoi atomi troppo rapidamente, impedendo loro di raggiungere quelle ottimali per una struttura cristallina sufficientemente regolare e quindi utilizzabile. Volendo fare un esempio nel campo della Fisica dello Stato Solido, è noto che producendo in questo modo errato i cristalli di silicio da utilizzare come rivelatori, si ottengono moltissime impurezze reticolari, le quali agiscono né più né meno come delle impurezze dannose, e rendono questi cristalli totalmente inutili allo scopo. L'annealing, invece, consiste in una fusione rapida accompagnata da un lento processo di raffreddamento, in modo da consentire agli atomi di assestarsi nel modo migliore ed ottenere un cristallo perfettamente strutturato. Rivedendo il fatto in termini di livelli energetici, si può riguardare il processo "rapido" come un metodo che, una volta intrappolato lo stato del sistema (inteso qui come insieme delle posizioni dei nodi reticolari) in un avvallamento della sua funzione di energia interna, si stabilizza nel minimo relativo di questo. Nel caso del procedimento di annealing, invece, si consente al sistema di passare da un avvallamento all'altro, in modo da cercare quello più profondo, il cui punto di minimo corrisponde al vero stato fondamentale del cristallo. Da questo paragone, è probabilmente già chiaro il parallelo con le reti di Hopfield, essendosi già proposto uno schema simile, osservando il processo di aggiornamento di tali sistemi alla luce delle variazioni della loro energia.

Adesso è scopo della presente discussione sfruttare i concetti della meccanica statistica in modo da riprodurre una versione virtuale di annealing per le reti neurali. Il processo consiste, in termini virtuali, nell'attendere che il sistema si ponga in equilibrio termico con il serbatoio di calore, il che avviene non necessariamente con passaggi atti a diminuire sempre l'energia, ma anche con qualche possibilità controllata di transizioni al rovescio, le quali non sono più vietate, come avviene nelle reti di Hopfield "tradizionali", ma solo altamente improbabili. Il passo successivo è quello di abbassare di poco la temperatura e aspettare il nuovo equilibrio, e così via. Con questo lavoro, il sistema passa da un avvallamento dell'energia interna all'altro, e man mano che la sua energia diminuisce, diventano sempre meno quelli, tra questi avvallamenti, che sono abbastanza profondi da poter essere "visitati". Questo ragionamento permette di stabilire che, quanto più il sistema è "freddo" abbastanza, tanto più è probabile che si trovi entro la buca energetica più profonda. A questo punto, lo si può costringere ad andare verso il minimo dell'avvallamento in cui giace, e si avrà una notevole probabilità che tale minimo sia quello assoluto.

L'unico scoglio da superare in tutto questo discorso è la necessità di definire una "temperatura" per le reti neurali, che non si può dedurre da proprietà energetiche fisicamente osservabili, in quanto l'energia della rete è una funzione informatica, non certo fisica. L'unico modo è fissare *ad hoc* una variabile di controllo, che viene decrementata regolarmente durante il processo di aggiornamento.

Il modello di base dell'annealing simulato sfrutta ancora i neuroni binari. Si immagini di indicare con l'indice k l'unità in fase di elaborazione. La sua attivazione potrà valere 0 od 1, per cui si possono calcolare le energie corrispondenti ad entrambe le situazioni, E_0 ed E_1, mantenendo tutte le altre attivazioni costanti. Conoscendo l'espressione dell'energia e tenendo conto che tutte le attivazioni della rete possono valere 0 o 1, è facile risalire subito alla differenza tra i due valori trovati. Nel caso di E_0, tutti gli addendi contenenti l'attivazione del neurone in studio spariscono, mentre gli stessi addendi avranno, in E_1, il valore del prodotto fra ogni peso sinaptico e l'attivazione dell'altro neurone con cui viene condiviso. In questo modo, tenendo conto che nell'energia compare due volte ogni termine, per la degenerazione di scambio rispetto agli indici, è possibile risalire alla seguente

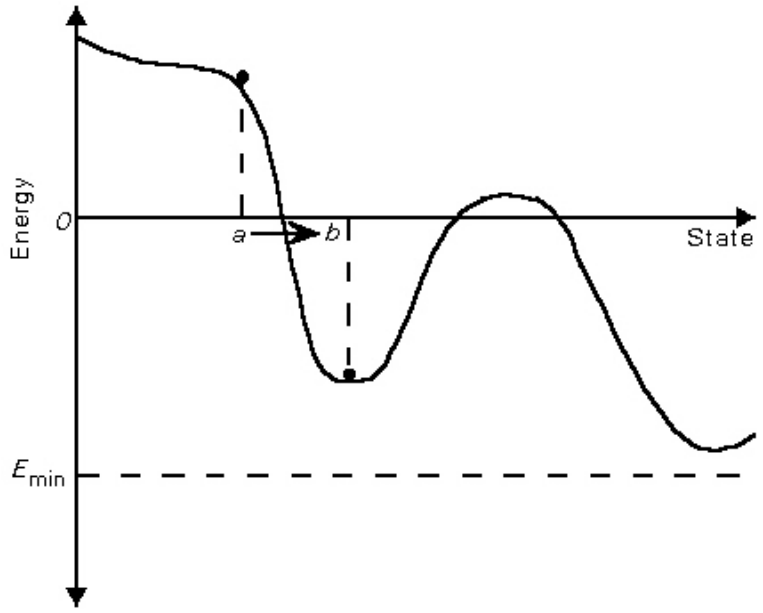

Figura 4.6: Tipico schema dell'andamento dell'energia di una rete di Hopfield. Notare i due minimi. Senza annealing, se lo stato iniziale fosse nel punto di ascissa a, quello finale sarebbe nel minimo relativo e quindi non corretto. Con l'annealing si dà la possibilità al punto di risalire la china a destra di b, in modo da finire nella buca di energia che ha il suo punto di minimo nel vero minimo globale della funzione.

conclusione:

$$E_0 - E_1 = \sum_i w_{ik} x_i = net_k \tag{4.13}$$

In questo caso, si comprende, l'input complessivo ricevuto non serve direttamente per dedurre la nuova attivazione, ma è la differenza tra l'energia del sistema nelle due possibili attivazioni del k-esimo neurone. Si deve immaginare, adesso, il sistema in un'ottica parziale, in cui tutte le attivazioni degli altri elementi è fissa, come anche la temperatura, per cui l'unico parametro da cui dipende l'energia, in questo momento, è l'attivazione del k-esimo neurone. Allora, la funzione di partizione di questo sistema è simile a quella del modello di Ising, in cui però si somma rispetto ad una sola componente. Dal momento che le possibili attivazioni sono due e valgono 1 e 0, si può esprimere questa funzione di partizione per esteso, e calcolare esattamente la probabilità di ottenere una a caso fra le energie E_0 ed E_1. Volendo, per esempio, sapere che probabilità ci sia di ottenere E_1, si ricava:

$$p(E_1) = \frac{e^{-\frac{E_1}{T}}}{e^{-\frac{E_1}{T}} + e^{-\frac{E_0}{T}}} = \frac{1}{1 + e^{-\frac{E_1 - E_0}{T}}} = \frac{1}{1 + e^{-\frac{net_k}{T}}} \tag{4.14}$$

Ciò che bisogna fare, allora, è assegnare al neurone in questione l'attivazione 1 con una probabilità data da quest'espressione. Tuttavia, l'aggiornamento deve consistere nel compiere una sola modifica alla volta per ogni neurone. Il modo di implementare questo concetto nel processo effettivo, è lo stesso che si usa quando si esegue un'estrazione col metodo Montecarlo di numeri distribuiti in base ad una funzione qualunque. Bisogna fare l'estrazione di un numero casuale tra 0 ed 1, e poi, se questo valore casuale sarà minore della probabilità (4.14), si assegnerà al neurone l'attivazione 1, altrimenti le si assegnerà 0. Naturalmente, più è piccolo net_k, più sarà difficile che l'attivazione estratta a caso valga 1.

Volendo, a questo punto, fare una piccola precisazione, il neurone usato in questa rete neurale, non esattamente è quello binario a soglia, ma si comporta piuttosto come quello che nel capitolo 2 si è chiamato "neurone termodinamico". Si comprende qui, allora, quale possa essere l'utilità di un neurone dal comportamento apparentemente così singolare.

Naturalmente, il processo di aggiornamento va ripetuto per tutti i neuroni allo stesso modo, anzi, per migliorare il parallelismo con le fluttuazioni statistiche del caso, sarebbe utile sorteggiare ogni volta l'unità da aggiornare. Il procedimento si compone di diversi cicli, in cui un ciclo consiste in un certo numero di passaggi del tipo di quello proposto, tutti alla stessa temperatura. Terminato il ciclo, la temperatura va diminuita leggermente e si ripete il lavoro. Naturalmente, per quanto numerosi possano essere i passaggi in un ciclo, non si può mai assicurare che tutte le unità vengano interessate ogni volta. Il modo di procedere, quindi, deve prevedere di compiere per ogni ciclo una quantità di revisioni delle attivazioni neurali pari ad un multiplo, anche se non troppo grande, della quantità di neuroni presente.

Ecco, per concludere e sintetizzare, uno schema dei passaggi da eseguire per operare l'aggiornamento di una rete di Hopfield col metodo dell'annealing simulato:

1. impostare a caso l'attivazione iniziale per ogni neurone;
2. selezionare *a caso* un'unità e calcolarne l'input complessivo;
3. calcolare il termine probabilistico p dalla (4.19);
4. estrarre un valore a caso $q \in [0, 1]$ ed assegnare al neurone l'attivazione 0 se $q > p$ e 1 in caso contrario;
5. ripetere i passaggi 1–4 per un ciclo di aggiornamenti a temperatura fissata T;
6. abbassare T e ripetere il passaggio 5 per un altro ciclo.

Generalmente, il criterio di decrescenza di T è definito dalla "scheda di annealing", la quale può prevedere una funzione monotona che decrementa T ad ogni ciclo, come anche di ripetere più cicli con la stessa temperatura, prima di diminuirla.

Quando il sistema giunge a temperature molto basse, naturalmente, nemmeno i possibili spostamenti nel senso in cui l'energia cresce consentono al sistema di sfuggire alla buca in cui si è localizzato. Quindi, in questo caso, viene raggiunto il minimo con un criterio simile a quello "tradizionale", ma notevolmente più lento, in quanto i passaggi casuali comportano l'avvicinamento in modo molto meno diretto. Quindi, se è possibile controllare il processo di aggiornamento, dopo aver portato la temperatura a valori piccoli, la si può porre uguale a zero, la qual cosa significa che, a quel punto si ritorna ad utilizzare il criterio di aggiornamento iniziale delle reti di Hopfield.

L'unico grande svantaggio di questo metodo è proprio la sua estrema lentezza, in quanto le numerose estrazioni a caso implicano lunghi tempi di stabilizzazione. In molti casi in cui si vuole una risposta estremamente pulita, si potrebbe dire che "il gioco vale la candela", ma non sempre è così.

Un altro discorso è che la rete con annealing simulato, proposta in questo modo, è molto efficace nel caso in cui la rete possegga degli esempi fisici sui quali viene addestrata, mentre non lo è altrettanto quando il problema è da definire a priori. Il passaggio successivo è, allora, quello di introdurre la teoria nota come MFT, da "Mean Field Theory", la quale produce una nuova evoluzione della rete di Hopfield, in cui si dà l'addio definitivo ai neuroni binari.

4.3.3 Reti MFT

La teoria del campo medio [Pet88] (banale traduzione di MFT) è un procedimento proposto nell'ottica di risolvere espressamente i problemi di ottimizzazione, cioè quelli in cui non è possibile definire esemplari per la rete neurale, ma solo criteri in base ai quali definire le soluzioni migliori. Ciò si traduce nella definizione di pesi sinaptici ottimali, a partire dai quali l'energia della rete, espressa nel modo consueto, si minimizza in corrispondenza delle configurazioni d'attivazione meglio rispondenti ai parametri valutativi prefissati.

Prova ne è che, stavolta, in quest'energia è prevista la possibilità di inserire espressioni più complicate della normale forma tipo-Ising, che contengono, con opportuni moltiplicatori di Lagrange, le eventuali condizioni, permettendo di risolvere qualunque problema di minimo vincolato. Sostanzialmente, quindi, l'energia possederà due contributi: il "costo", che è la parte dipendente dai pesi e che costituisce l'energia determinata a partire dello stato effettivo del sistema, ed il "vincolo" che invece contribuisce in base alla rispondenza dello stato con i parametri predefiniti di controllo, e tende ad innalzare l'energia se la corrispondenza peggiora. In questo modo, la stabilizzazione della rete è un compromesso tra la configurazione di "costo" minimo e quella di maggiore adattamento al "vincolo", in quanto vengono scoraggiati i passaggi che, seppure decrementassero di molto il "costo", conducono ad un notevole aumento del "vincolo". Naturalmente, se l'energia deve decrescere andando verso la stabilizzazione, la sua espressione deve essere:

$$E = a \cdot \text{costo} + b \cdot \text{vincolo}$$

con a negativo e b positivo.

Un particolare vantaggio che questo metodo comporta è il fatto che riconduce i calcoli ad essere deterministici, nel senso che la determinazione della nuova attivazione di un'unità si calcola a partire da una funzione definita delle altre e dei pesi, senza più processi casuali, il cui influsso viene considerato in media, e comporta l'unica necessità di usare attivazioni reali, proprio perché valori medi di entità che possono valere 0 o 1. Inoltre, il processo è tale da consentire, rispetto all'annealing, maggiore rapidità e anche minore sforzo per un elaboratore, il che aiuta molto nella soluzione al problema della "scalabilità" di un modello neurale. Questo problema, in sintesi, sorge quando un modello diventa improponibile per situazioni realistiche, per il fatto che se i neuroni divengono troppo numerosi, i tempi di calcolo e le richieste di supporto di memoria non sono facili da soddisfare. È chiaro che se una rete neurale soffre di questo problema, non ha molto valore pratico, ma può fungere soltanto da modello didattico, risolvendo quelli che vengono denominati "toy problems", ossia problemi fittizi molto semplici, che comunque non raggiungono la complessità delle questioni di interesse concreto.

Per quanto si sia detto che l'energia può assumere forme complesse rispetto a quella consueta, si può benissimo supporre che tale complessità conduca sempre alla forma nota, in cui i pesi contengono nella loro definizione numerica tutti i parametri di cui serve tenere conto, o almeno questa sarà l'immagine di partenza, dove nulla toglie che i moltiplicatori di Lagrange siano inseriti nelle espressioni funzionali dei pesi sinaptici.

Il primo passo della teoria del campo medio consiste nell'esprimere la funzione di partizione di Boltzmann introducendo una definizione diversa, che sfrutta variabili continue. Esse sono raggruppate in vettori di dimensione N, ove questo è il numero dei neuroni nella rete, che, a differenza di $\mathbf{S}$ hanno componenti reali e le si rappresenterà come $\mathbf{U}$ e $\mathbf{V}$. In

base a queste variabili, si può definire la seguente funzione:

$$G(\mathbf{V}, \mathbf{U}, T) = \frac{E(\mathbf{V})}{T} + \sum_{i=1}^{N} [U_i V_i - \log(\cosh U_i)] \tag{4.15}$$

tale che si possa scrivere:

$$Z = \int_{-\infty}^{\infty} dV_1 \cdots \int_{-\infty}^{\infty} dV_N \int_{-i\infty}^{i\infty} dU_1 \cdots \int_{-i\infty}^{i\infty} dU_N e^{-G(\mathbf{U},\mathbf{V},T)} \tag{4.16}$$

Mediante un'espansione col metodo del punto sella (per i dettagli si veda l'appendice A di [Pet88]), è possibile compiere un'approssimazione di campo medio, che tiene in considerazione l'energia media per neurone e la approssima ad una serie troncata di Taylor attorno, appunto, al suo punto sella. Ciò che si fa, almeno in una spiegazione fisica, è la scelta di imporre, piuttosto che la banale minimizzazione dell'energia, un bilanciamento tra energia ed entropia del sistema. Le equazioni che vengono fuori da queste considerazioni sono le seguenti:

$$\begin{cases} V_1 = \tanh U_I \\ \frac{\partial E(\mathbf{V})}{\partial V_i} = -\frac{U_i}{T} \end{cases} \tag{4.17}$$

Da questi risultati deriva quindi una regola d'aggiornamento che, in pratica, opera un compromesso tra quella di Hopfield e quella dell'annealing simulato, in quanto calcola l'input complessivo come nella prima e vi applica la funzione della seconda. La differenza è che stavolta, il risultato della funzione è direttamente l'attivazione del neurone:

$$V_i = \tanh\left(\frac{1}{T}\sum_i T_{ij} V_j\right) \tag{4.18}$$

Le equazioni di questo genere sono quelle che, per la precisione, si definiscono "equazioni di campo medio". Si può mostrare (si veda rif. 1 e 12 in [Pet88]) che queste variabili continue V_i rappresentano una media sulla temperatura delle attivazioni dell'i-esimo neurone. Si noti come la funzione d'attivazione, a parte differenze sull'estensione del codominio, è la stessa che si è proposta nella definizione della variante continua della rete di Hopfield.

Una differenza sostanziale rispetto al metodo dell'annealing simulato è che in questo procedimento la temperatura non va diminuita, ma fissata ad un valore opportuno e mai toccata. È anche chiaro che se, come si è detto, si vuole complicare la forma dell'energia con moltiplicatori di Lagrange, della qual cosa si vedrà un esempio nell'applicazione proposta in questa tesi, allora è importantissimo un opportuno processo di tuning delle variabili libere, perché una regolazione sbagliata porta la rete a conclusioni del tutto inaccettabili.

È questo il risultato finale cui si intendeva pervenire. Adesso, per applicare al problema in esame tale algoritmo, basterà definire i pesi sinaptici della rete in modo opportuno, e mappare il problema stesso in termini neurali. Il modo in cui ciò si realizza sarà proposto nel prossimo capitolo, mentre adesso è utile proseguire questo con esempi di implementazione dei modelli di rete neurale propostivi.

4.4 Esempi di implementazione

4.4.1 Rete discreta di Hopfield

Come fatto nel caso del percettrone, il problema di fondo è conoscere le caratteristiche di cui è importante tener conto per creare una struttura dati del neurone di Hopfield. In

questo caso, esse sono l'attivazione, binaria, e la soglia, reale. Allora una possibile struttura dati da usare è la seguente:

```
STRUCTURE Neuron_Type {
   BOOL Act;
   REAL Thresh;
}
```

Come nell'esempio del percettrone, si dichiara un array di elementi di questo tipo, ciascuno dei quali è un neurone. Si porrà in seguito che tale array si sia chiamato `Unit[]`. In questo schema, il peso sinaptico di connessione tra ogni coppia di unità può essere rappresentato da un array reale a due indici. Tuttavia, dal momento che in molti casi i neuroni sono molti, è il caso di fare un'analisi della RAM che si andrebbe ad occupare, ed ottimizzare per quanto possibile il criterio di "data storage" per impegnare al minimo la memoria del computer. La struttura dei pesi memorizzati come una matrice è molto semplice, ma comporta un doppione per ogni peso. Considerando che in genere un valore booleano ha dimensione di un byte e una variabile reale di almeno 4, per N neuroni si ottiene un'occupazione di RAM uguale a

$$S = N + 4N + 4N^2 = 5N + 4N^2$$

Se per, ad esempio, 20 neuroni (dove 20 è un numero contenuto), tale occupazione è di $5 \cdot 20 + 4 \cdot 20^2 = 1700$ byte, ossia 1.7 kbyte, nel caso più realistico di 50000 neuroni, quale è quello che sarà in esame nello specifico di questo lavoro di tesi, si avrebbe $S > 10^{10}$ byte in tutto, ossia più di 10 Gb di memoria, che non si hanno a disposizione nemmeno nel più potente dei computer disponibili per uso comune. Il modo migliore di minimizzare questo lavoro è quello di eliminare le ripetizioni, ma a questo punto, ogni peso deve tenere conto, oltre che del suo valore, anche dei due elementi che connette. Si può adoperare una struttura come questa:

```
STRUCTURE Weight_Type {
   REAL Value;
   INT u1;
   INT u2;
}
```

Il motivo per cui si pongono due interi è quello di snellire la ricerca senza appesantire la memoria. Infatti, si presuppone, in questo schema, di aver definito l'array dei neuroni, in modo che si possa fare riferimento all'indice di questi per localizzarli in memoria. Naturalmente, anche i pesi costituiranno un array, ma monodimensionale stavolta, per cui si avrà una variabile - la si chiamerà `Weight[]` - del tipo `Weight_Type`. In questo modo, per trovare, ad esempio, la prima unità cui si riferisce l'i-esimo peso, basta cercare l'elemento `Unit[Weight[i].u1]`, accelerando le ricerche a scapito di una minima perdita di comprensibilità del listato. Naturalmente, esisteranno nel programma due variabili intere, diciamo `Unit_n` e `Weight_n` che contengono le dimensioni degli array di neuroni e pesi. Con questo accorgimento, tra l'altro, si possono evitare, oltre ai doppioni, gli inutili termini in cui un neurone comunica con se stesso, che sono N anch'essi. La dimensione della struttura `Weight_Type` è adesso quella di tre campi di dimensione 4, però avremo un numero di elementi uguale alle combinazioni di N elementi presi a due a due, senza ripetizione. In tutto, se la dimensione della struttura `Neuron_Type` è di 3 byte (un reale e un booleano), allora la memoria totale occupata sarà:

$$S = 3N + 4\binom{N}{2} = 3N + 2N(N-1) = 2N^2 + N \tag{4.19}$$

Si osservi in figura 4.7 la differenza tra i valori di occupazione della RAM nei due casi.

Si può notare dal grafico come in genere l'array semplice con la forma proposta comporti un impiego di circa la metà della RAM necessaria per l'altro.

In questo schema strutturale, la procedura per aggiornare l'attivazione di un elemento, contraddistinto con l'indice k, con un aggiornamento seriale, può funzionare in questo modo:

```
SUB Serial_Update()
   REAL Input = 0.0;
   INT i, k;
   FOR (k, 0, Unit_n)
      FOR (i, 0, Weight_n)
         IF((Weight[i].u1 = k AND Unit[Weight[i].u2]) OR_
         (Weight[i].u1 = k AND Unit[Weight[i].u2]))
            Input = Input + Weight[i].Value;
         EBD IF
      END IF
      Unit[k].Act = (Input > Unit[k].Thresh);
   END FOR
END SUB
```

Il funzionamento è abbastanza semplice: per ogni variabile si imposta all'inizio un valore reale a 0, e, per ogni volta che, tra i pesi, ne viene trovato uno tale che si verifichino contemporaneamente le seguenti condizioni:

- uno dei campi `u1` o `u2` è l'indice dell'unità in fase d'aggiornamento;
- l'altro campo indica un neurone attivo;

in questo caso, all'input viene sommato il peso, dato che l'attivazione è 1 e quindi moltiplicarla per esteso non serve. Finito il ciclo su tutto l'array dei pesi, si confronta l'input con la soglia e si dà l'attivazione 1 se la soglia è minore dell'input, altrimenti 0. Il modo in cui quest'istruzione viene espressa è quello di dare all'attivazione il valore dell'espressione di confronto, pratica diffusa nel caso in cui una variabile booleana deve il suo valore ad una disuguaglianza. Con lo stesso tipo di ragionamento, nella complessa istruzione `IF` proposta (e splittata in due righe mediante l'underscore), una parte della condizione è rappresentata dalla semplice attivazione del neurone, che è appunto un valore che può essere "vero" o "falso". A parte queste osservazioni, il resto del listato sarà probabilmente abbastanza chiaro, alla luce delle convenzioni di cui all'appendice A.

Volendo produrre una funzione che determini un aggiornamento parallelo, si potrebbe usare una funzione come la seguente:

```
SUB Parallel_Update()
   REAL Input[Unit_n] = 0.0;
   INT i;
   FOR (i, 0, Weight_n)
      IF ((Unit[Weight[i].u1])
         Input[Weight[i].u2]=Input[Weight[i].u2]+Weight[i].Value;
      END IF
      IF ((Unit[Weight[i].u2])
         Input[Weight[i].u1]=Input[Weight[i].u1]+Weight[i].Value;
```

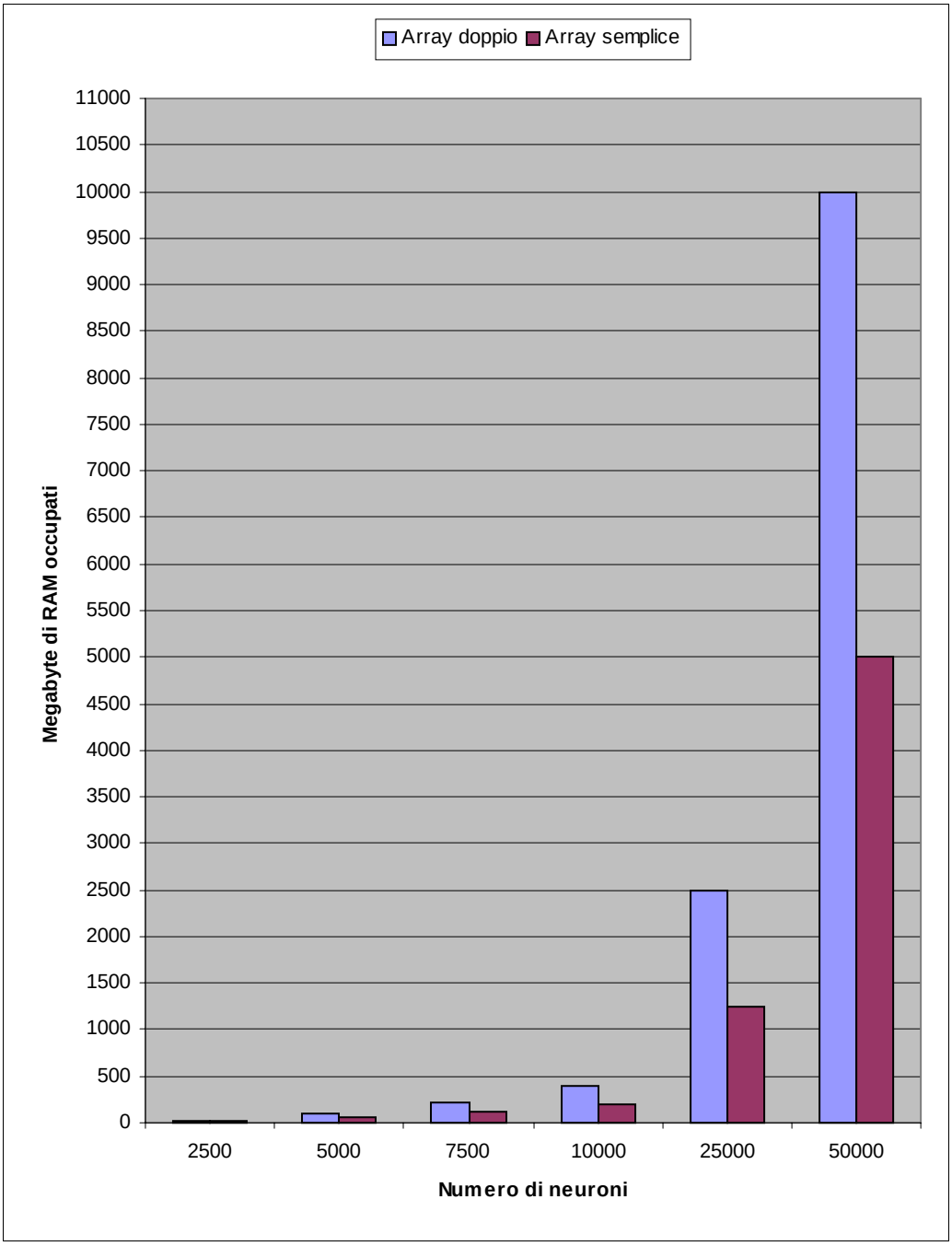

Figura 4.7: Confronto tra metodo ad array bisimensionale e monodimensionale in termini di occupazione di RAM in una rete neurale simulata al computer.

```
      END IF
   END FOR
   FOR (i, 0, Unit_n)
      Unit[i].Act = (Input[i] > Unit[i].Thresh);
   END FOR
END SUB
```

In questa funzione, basta scorrere su tutti i pesi, e definire una variabile di incremento per ciascun neurone. Per ogni peso, se uno dei due neuroni è attivo, contribuirà all'input dell'altro con un valore uguale al peso sinaptico memorizzato, per cui tale valore viene aggiunto all'input totale dell'altro neurone. Se invece esso è spento, non porta alcun contributo. In questo modo, vengono incrementati gradualmente tutti gli input, senza modificare le attivazioni degli elementi. Quando si è terminato questo lavoro, vengono confrontati gli input complessivi ottenuti con le soglie dei diversi neuroni, aggiornando le attivazioni in modo opportuno. Naturalmente, nel caso in cui si giungesse ad uno stato oscillante, questa routine sarebbe senza fine, quindi occorrono opportune strutture di controllo. Inoltre, serve dappertutto un controllo che consenta di conoscere quando il ciclo va completato. A questo scopo, basta creare un array temporaneo, in cui depositare le attivazioni precedenti di tutte le unità e confrontarlo con quelle aggiornate. Nel caso in cui i due array coincidessero, il ciclo di aggiornamento si può considerare concluso. Se si tiene memoria anche dell'attivazione di due cicli prima, si può confrontare l'attivazione nuova con le ultime due, in modo che, se risultano cambiamenti rispetto all'ultima ma non rispetto alla penultima, si può concludere di essere entrati in un'oscillazione tra stati ed uscire comunque.

Quanto all'addestramento, quando servisse, è molto semplice implementarne un esempio con la regola di Hebb. Si suppone, ancora una volta, di avere definito banalmente una subroutine per inizializzare i valori delle attivazioni, per la qual si usi il nome `Initialize(Example[i])`, ove l'argomento è il vettore di esempio. La subroutine può essere descritta dal seguente algoritmo:

```
SUB LearnCycle(Real Eta)
   INT i, j;
   FOR (i, 0, Examples_n)
      Initialize(Example[i]);
      FOR (j, 0, Weight_n)
         IF (Unit[Weight[j].u1].Act XOR Unit[Weight[j].u1].Act)
            Weight[i].Value = Weight[i].Value - Eta;
         ELSE
            Weight[i].Value = Weight[i].Value + Eta;
         END IF
      END FOR
   END FOR
END SUB
```

Si sarà notato che in realtà la subroutine non aggiunge al peso il prodotto delle attivazioni, ma solo il termine di controllo `Eta` proposto in argomento. Il motivo è che la regola di Hebb deve rinforzare la connessione tra due unità concordi, ma qui, essendo i neuroni ad attivazione booleana, "vero" corrisponde sì ad 1, ma "falso", generalmente, corrisponde a 0, per cui il contributo incrementante dovuto ai neuroni spenti sparirebbe. In questo modo, invece, si controlla direttamente se i due neuroni sono concordi o discordi, e nel primo caso si incrementa il peso, mentre nel secondo si decrementa. L'istruzione `XOR`, si

ricordi, restituisce "vero" o "falso" a seconda che i due argomenti siano dello stesso tipo o meno.

4.4.2 Rete di Hopfield con annealing

Proporre un criterio di annealing simulato, in realtà non complica molto le cose, circa la rete neurale simulata. È possibile, infatti, mantenere le stesse strutture di dati e le stesse variabili. L'unica cosa da modificare è la procedura di aggiornamento.

Tenendo conto di come avviene il processo, naturalmente, il criterio parallelo non è applicabile, mentre si può usare, come punto di partenza, la stessa subroutine di aggiornamento seriale proposta precedentemente, operando le necessarie modifiche e ponendo di avere definito le funzioni matematiche opportune stabilite, ossia l'esponenziale, che si indicherà con `EXP()` e il generatore di numeri casuali tra 0 ed 1, che sarà rappresentato da `RND()`:

```
SUB Annealing_Update(REAL Temp)
   REAL Input = 0.0, Prob, Casual;
   INT i, k;
   FOR (k, 0, Unit_n)
      FOR (i, 0, Weight_n)
         IF ((Weight[i].u1 = k AND Unit[Weight[i].u2]) OR_
         ( Weight[i].u1 = k AND Unit[Weight[i].u2]))
            Input = Input + Weight[i].Value;
         END IF
      END FOR
      Prob = 1 / (1 + EXP(-Input / Temp));
      Casual = RND();
      Unit[k].Act = (Casual > Prob);
   END FOR
END SUB
```

Naturalmente, in questa subroutine bisogna specificare il parametro di temperatura, e alcune nuove variabili, in cui si memorizza il termine probabilistico e il numero casuale. I nomi sono abbastanza evidenti per comprendere quali siano i termini adibiti a quest'uso.

Non è il caso di perdersi nelle tecniche di addestramento di questo tipo di rete, le quali prevedono concetti statistici che, oltre a minimizzare l'energia, intendono massimizzare l'entropia del sistema. Si è visto come l'analogia del modello di Ising permette infatti di determinare anche una "entropia" di una rete neurale. È possibile dimostrare [Free91] che quando l'entropia informatica è massima, le realizzazioni del sistema sono tutte equiprobabili, ed è questo il caso cui si vuole giungere, nell'intento di avere un certo numero di esemplari, i quali debbono essere tutti, inizialmente, dei buoni candidati all'estrazione e solo dopo una serie di aggiornamenti viene scelto il migliore, anche in base alla posizione iniziale del vettore d'attivazione. Questo è un discorso abbastanza specifico che non è il caso di sviluppare oltre, per non perdere di vista i concetti di interesse principale nel presente studio.

Essendo il metodo MFT quello che viene implementato nel programma realizzato per il calcolo specifico di cui si parlerà in seguito, non è il caso di dilungarsi in listati esemplificativi. Si passerà invece a presentare i suoi aspetti implementativi quando verrà proposto l'algoritmo effettivamente utilizzato. Si procederà in due passaggi: prima una definizione del metodo generico, e poi il suo adattamento al caso in questione. Per tutto questo, si

rimanda al capitolo successivo, dopo la breve introduzione sui mezzi fisici che producono i dati da analizzare.

Capitolo 5

Ricostruzione di tracce nella TPC di ALICE con l'algoritmo MFT-neurale. Mappatura del problema e risultati dalle simulazioni

Dopo aver introdotto, nel capitolo scorso, i concetti di base, è adesso il momento di proporre l'applicazione specifica per la ricostruzione delle tracce. Prima di procedere con la descrizione dettagliata del metodo e dell'algoritmo con il quale lo si è implementato, tuttavia, è necessario avere chiaro che tipo di dati si ottengono da un evento simulato di ALICE. Ciò implica la necessità di illustrare brevemente il rivelatore da cui si estraggono i dati che vengono poi passati al programma di ricostruzione. Ciò serve a comprendere perché il problema verrà mappato in un certo modo, e anche le dimensioni e le complicazioni dell'operazione, in termini dei mezzi utilizzati per eseguirla, e della mole di dati da gestire.

5.1 La TPC di ALICE

Non è questa una sede in cui si intenda descrivere in pieno dettaglio tutte le caratteristiche dell'esperimento in questione, né è interesse di questa tesi illustrarne i diversi aspetti teorici e tecnici. L'unico scopo di questo paragrafo è quello di mostrare i caratteri salienti del rivelatore che fornisce i dati sui quali si applica il metodo in studio. Nel capitolo 1, infatti, si è evidenziato come l'intero apparato sperimentale di ALICE preveda diversi elementi, ciascuno dei quali è studiato per fornire un preciso tipo di contributo alla rivelazione, partecipando cosí alla ricostruzione dell'evento.

Dal momento che il problema è quello di ricostruire le tracce delle particelle, il rivelatore di interesse maggiore, perché fornisce il contributo principale al processo in questione, è la TPC. Per questo la discussione verterà essenzialmente su tale elemento.

5.1.1 Descrizione generale

La figura 5.1 mostra un prospetto della TPC di ALICE.

La TPC di ALICE è essenzialmente costituita da una struttura cilindrica, disposta in modo che i proiettili collidano lungo il suo asse di simmetria rotazionale, indicato con

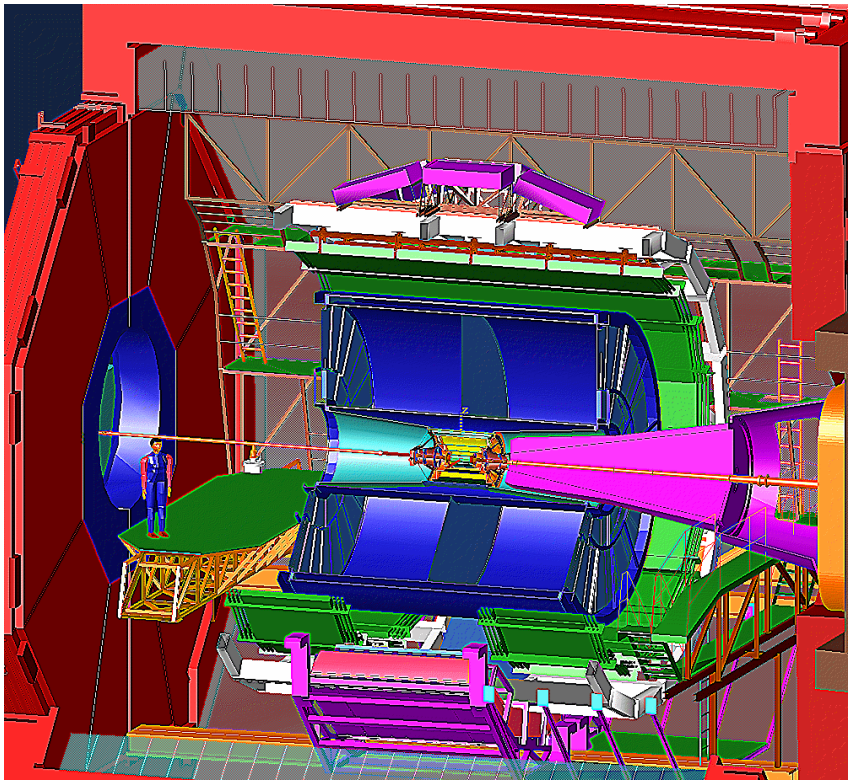

Figura 5.1: Prospetto della TPC di ALICE (la regione cilindrica centrale), integrato nella struttura di ALICE e con un confronto rispetto alla dimensione di un uomo.

z in tutta la tesi. Più precisamente, la collisione avviene nel centro di simmetria della struttura. Lungo quest'asse giace anche un debole campo magnetico, di modulo pari a 0.2 T, che ha lo scopo di costringere le particelle cariche, le uniche "visibili" a questo dispositivo, a percorrere traiettorie più o meno curve, a secondo del loro impulso, all'interno del rivelatore. Il raggio ed il passo di queste traiettorie, che possono essere assimilate, in prima approssimazione, a delle eliche, dipendono, oltre che dalla carica di ogni particella, dalle tre componenti del suo vettore impulso. Quella ottenuta dalla proiezione sul piano xy, che si suole indicare come impulso trasverso (p_t), ne stabilisce il raggio sullo stesso piano, mentre la componente z stabilisce il passo lungo l'omonimo asse. In questo modo, dallo studio della curvatura di una traccia, si può risalire all'impulso della stessa. Dal momento che, poi, la TPC è in grado di misurare anche la perdita specifica di energia della particella, è possibile pure procedere ad un'analisi che conduca al riconoscimento della specie della particella rivelata.

Come è noto, la TPC si annovera nella grande classe dei rivelatori a ionizzazione in mezzo gassoso, della quale costituisce uno degli elementi più complessi ed efficienti, essendo in grado, unica tra tutti gli esempi standard di essa, di restituire diversi punti per ogni particella che la attraversa, consentendo una notevole efficienza nella ricostruzione del percorso seguito da ciascuna. Una simile proprietà nasce dal principio di funzionamento, che, come è noto, sfrutta, per la misura della posizione nel piano trasversale all'asse z, una filiera suddivisa in settori (row), che restituiscono una coordinata assimilabile al modulo polare della posizione, in cui l'anomalia viene specificata mediante la suddivisione di ogni filo in diversi sub-settori (pad), delimitati da strutture sensibili che trasformano ogni filo in una sequenza di piccoli contatori proporzionali, in modo che, con un metodo come quello

del baricentro, si ricostruisce, per ogni segnale, la coppia di coordinate polari del punto in cui è stato generato. Quanto alla coordinata z, invece, la si valuta dalla misura del tempo di ritardo nell'arrivo del segnale registrato in ogni pad-row attivato. Nell'ipotesi migliore, ogni riga di pad (o pad-row) riceve un segnale dovuto a ciascuna particella, con il risultato che il suo passaggio comporta l'acquisizione di diversi punti, in cui la determinazione del pad colpito consente di localizzare la traccia nel piano trasversale ad ogni valore del modulo previsto, mentre il tempo di deriva che condiziona il ritardo del segnale individua la posizione dello stesso punto sull'asse z.

Il piano perpendicolare alla direzione di incidenza relativa dei fasci, e passante per il centro di simmetria del rivelatore, contiene un elettrodo centrale, a potenziale più negativo di quello posto sulle filiere dove si trovano i pad-row, allo scopo di accelerare gli elettroni prodotti dalla ionizzazione verso le estremità della struttura dove i pad-row stessi si trovano. Anzi, è bene sottolineare come, per la struttura simmetrica della TPC e per il fatto che essa opera su una collisione proiettile-proiettile, essa possegga un'area sensibile ad entrambe le estremità, in modo da consentire la rivelazione dei residui emessi tanto in avanti quanto all'indietro, rispetto all'orientamento dell'asse z.

Ciò che interessa principalmente illustrare adesso, come si è introdotto nel capitolo 1, sono le previsioni e le conseguenti richieste di efficienza in base al tipo di fenomeni che si attende di osservare in una collisione Pb-Pb all'energia prevista a cui l'acceleratore LHC dovrà lavorare. Ciò restituirà i parametri sui quali andrà modellata la rete neurale, per poter rispondere in modo efficiente al problema.

5.1.2 Caratteri specifici dell'esperimento ALICE

Anzitutto, è bene dire che la TPC di ALICE costituisce il dispositivo principale per la ricostruzione delle tracce, ma suo compito è anche fornire una misura della perdita specifica di energia di ogni particella all'interno del suo volume sensibile. Entrambe le misure sono necessarie per poter procedere all'identificazione di ogni oggetto fisico rivelato. Ricordando i grafici proposti nel capitolo 1, si comprende per quali valori di pseudorapidità e di impulso traverso la TPC è stata progettata:

- $|\eta| < 0.9$;
- $p_t < 10$ GeV/c.

Oltre a questo, bisogna tenere in considerazione la frequenza di interazioni Pb-Pb in LHC, che è di circa 8000 collisioni al secondo, delle quali circa 1000 centrali (quelle di maggior interesse), ossia con parametro d'impatto minore di 5 fermi (si ricorda che la somma dei raggi dei due ioni Pb che collidono è pari a circa 14 fermi). Con questi numeri, le previsioni dei modelli teorici forniscono una molteplicità di circa 8000 particelle primarie cariche per unità di pseudorapidità nella regione misurata, che possono arrivare a diventare più di 20000 in seguito ai decadimenti ed alla produzione di raggi δ dovuta alla ionizzazione prodotta dalle particelle primarie nel gas che riempie il volume sensibile del rivelatore. Tale molteplicità comporta, dalle simulazioni effettuate, un'occupazione media, per evento, del 40% dei pad-row più interni e di circa il 15% dei più esterni (per occupazione si definisce il rapporto tra il numero di elementi sensibili attivati in un evento ed il totale). Numeri del genere sono abbastanza al di sopra della maggior parte delle molteplicità generalmente prodotte in esperimenti dello stesso genere. Comportano, quindi, una notevole sfida per la realizzazione di un rivelatore che risponda con un potere risolutivo ed un'efficienza capaci

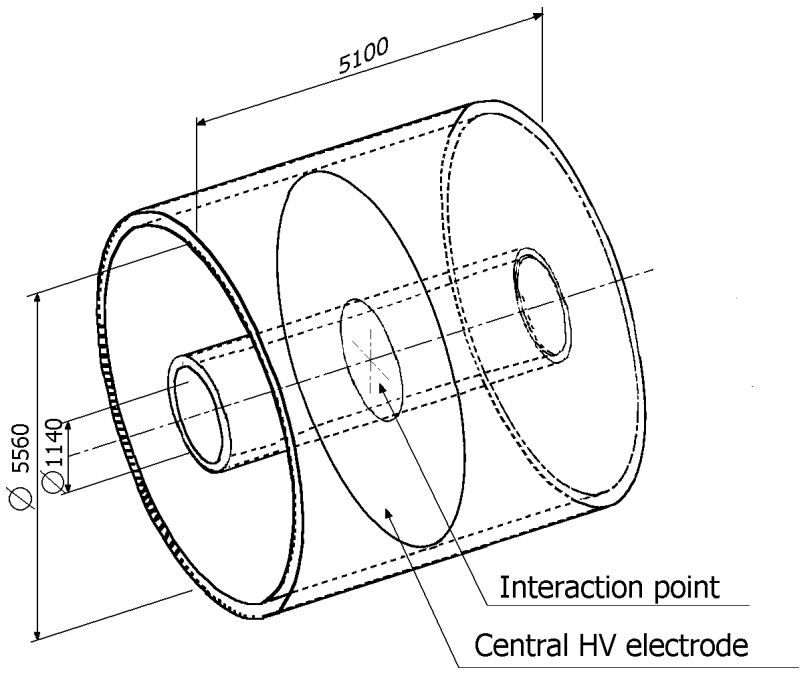

Figura 5.2: Schema geometrico della TPC, con alcune dimensioni specifiche per quella utilizzata in ALICE. Il volume sensibile è contenuto tra il cilindro più interno e quello più esterno.

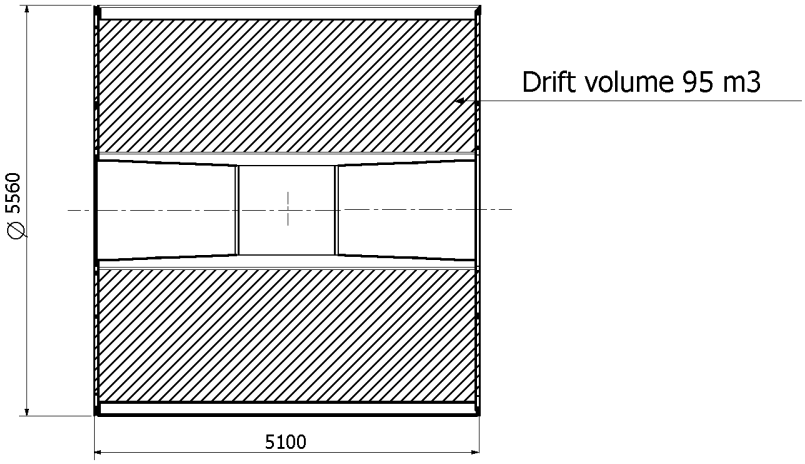

Figura 5.3: Prospetto laterale della TPC. La regione tratteggiata è la parte sensibile del rivelatore.

di gestirli in modo soddisfacente. Nell'ottica del tracciamento neurale, chiaramente, si dovrà fare in modo che la precisione raggiunta tecnicamente nel rivelatore, sia mantenuta nella ricostruzione dei dati che da esso provengono.

Ecco alcuni requisiti dedotti dalle previsioni illustrate finora, per quanto riguarda la rivelazione degli adroni:

- necessità di raggiungere una risoluzione di traccia di almeno 5 MeV/c;
- risoluzione energetica nella misura di dE/dx dell'8%;
- notevole efficienza nel *matching* con gli altri dispositivi che contribuiscono a fornire i dati per la ricostruzione di tracce: si parla del 90%;

Per la rivelazione dei leptoni, invece, questi sono i principali requisiti:

- efficienza di tracciamento maggiore del 90% per impulsi trasversi maggiori di 1 GeV/c;
- risoluzione dell'ordine del 2.5% per impulsi di circa 4 GeV/c, il che è necessario per una buona risoluzione nella misura delle masse, a sua volta fondamentale per l'identificazione delle particelle;
- risoluzione energetica nella misura di dE/dx dell'8%;
- capacità di sopportare un rate di almeno 200 Hz.

Queste richieste fisiche sono state soddisfatte nel modo migliore possibile, mediante una opportuna disposizione degli elementi strutturali essenziali. Oltre alla regione spaziale di accettanza, che è un cilindro il cui cerchio di base misura 5 metri di diametro e altrettanto in altezza totale, posta in modo da concordare con la regione di pseudorapidità esplorata dagli altri rivelatori del sistema, ecco le altre specifiche essenziali, che forniscono un'idea delle performance che tale TPC può raggiungere:

- "material budget" minimo possibile, in modo da rendere accettabile lo scattering multiplo e la produzione di particelle secondarie non dovute a decadimenti direttamente connessi con l'interazione da studiare;
- potenziale dell'elettrodo centrale di circa 100 kV, che è un valore ottimale per una buona resa con i gas utilizzati (si veda, in proposito, il prossimo punto). Ad esso si aggiungono delle filiere che degradano il campo in modo da mantenerlo uniforme fino alle estremità della camera, anche nella regione più periferica;
- utilizzo di una miscela gassosa che consenta di gestire una grande molteplicità, e quindi sostanze con una costante di diffusione bassa, che minimizzi il rischio di scattering multiplo: le preferenze sono cadute su miscele di Ar o Kr e CO_2 o CH_4;
- 570000 pad di lettura, per rendere il fattore di occupazione minimo possibile, comparato alle dimensioni dell'evento ed al tempo necessario alla codifica dell'informazione, con superfici crescenti man mano che si va dall'interno verso l'esterno (è chiaro che maggiore necessità di precisione c'è nelle regioni interne, in cui la densità di segnali è più elevata).

Per ogni specifica più dettagliata, si rimanda il lettore interessato al Techincal Proposal dell'esperimento ALICE [ALICE-TP] ed, in particolare, al Technical Design Report della TPC [TDR-TPC].

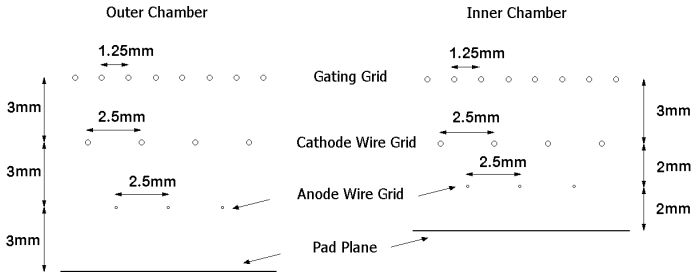

Figura 5.4: Distanze tra le diverse filiere delle due camere che costituiscono la TPC di ALICE.

5.1.3 Sistema di *read-out* della TPC

In questo sotto-paragrafo si intende illustrare alcuni aspetti delle strutture atte alla lettura ed alla codifica dei dati raccolti dal rivelatore e che sono poste alle estremità della camera che la TPC costituisce, sulle basi del cilindro (si vedano le figure). Ciò serve ad avere un'idea delle distanze tipiche tra i punti della traiettoria ricostruiti che poi l'algoritmo neurale andrà a leggere. Purtroppo, a tutt'oggi, la geometria, in termini di numero di elementi sensibili, non è del tutto definita, e quindi si proporrà il tipo di TPC sul quale si è lavorato, con la premessa che tutto il discorso, in termini di valutazioni numeriche e misure, è passibile di aggiornamenti futuri.

Ribadendo che il problema è quello di gestire molteplicità enormi, con una risoluzione soddisfacente tanto nella determinazione dell'impulso quanto in quella della perdita specifica di energia, si capisce come il principale scopo della TPC di ALICE sia quello di poter sopportare rate molto elevati di rivelazione e manifestare una buona capacità di separazione delle tracce. Per questo motivo si è scelto un sistema di lettura basato sulla tecnologia convenzionale delle multi-wire proportional chambers (MWPC).

Naturalmente, un obiettivo primario è stato quello di coprire la regione di spazio occupata nel modo migliore, evitando il più possibile le cosiddette zone morte. Tuttavia, non si può fare a meno di averne alcune tra i diversi settori. Infatti, la TPC è costituita da 18 settori trapezoidali disposti a raggiera, in modo da coprire l'intera dinamica angolare azimutale. Gli spessori morti sono le minime intercapedini tra due settori contigui, che comunque sono minimizzate il più possibile, mediante opportuni sistemi di montaggio.

Altro parametro di cui va tenuto conto è il fatto che l'addensamento molto intenso nella regione interna, ha reso necessario suddividere l'intero rivelatore in due settori, trattati in modo da avere diversi parametri di sensibilità ed efficienza. Il settore più interno e quello più esterno sono separati da uno spessore radiale, che si aggiunge allo spazio non sensibile, di circa 3 cm. Naturalmente, ciò comporta che l'ultimo punto registrato, per ogni traccia, nel settore interno, disti dal primo rilevato nel settore esterno, più della quantità di cui sono separati, in media, due punti rilevati nello stesso settore. Di questo bisognerà tenere conto nelle condizioni da imporre alla rete neurale, in quanto la distanza tra due elementi è un parametro che gioca un ruolo importante nel peso sinaptico eccitativo. In base a questa costruzione, ed alle dimensioni dell'intera struttura, illustrate nelle figure, risulta che l'area sensibile del rivelatore consente di rivelare punti di modulo radiale compreso tra 84.1 e 132.1 cm nella camera interna, e tra 134.6 e 246.6 cm nella camera esterna. Esiste, quindi, un gap di 2.5 cm tra le due, che è la minima distanza tra l'ultimo punto trovato nella camera interna ed il primo rivelato in quella esterna.

La principale differenza tra le due camere è illustrata dalla figura 5.4: Come si vede,

esistono tre filiere: la prima è quella della griglia, che serve agli scopi consueti di isolare la regione di deriva da quella di moltiplicazione, per evitare di osservare valanghe prodotte in diversi punti ed allo scopo di evitare che gli ioni prodotti ritornino nella regione di deriva, causando disturbi nei segnali ricevuti. Poi si trova la filiera catodica, dalla quale si estrae il segnale, e quella anodica, necessaria a completare lo schema, ma non usata per la lettura di dati. Per evitare un eccessivo accumulo di carica per unità di lunghezza di ciascun filo di lettura, si è stabilito di porre tali fili ad una distanza piccola fra loro: 2.5 cm separano ogni coppia contigua. Inoltre, dal momento che i fili anodici non vengono usati per raccogliere dati, non si rivelano necessari i "field wires", ossia i fili che servono a mantenere uniforme il campo elettrico nelle vicinanze di ogni anodo, in modo da guidare la deriva delle cariche. L'assenza di questi elementi, poi, garantisce una minor perdita di segnale e riduce l'intensità delle forze elettrostatiche che deformano i fili stessi, effetto negativo che va sempre minimizzato. L'unica conseguenza è la necessità di avere un potenziale più alto tra gli elettrodi della camera per ottenere il giusto fattore di moltiplicazione, o "gas gain".

Come è stato illustrato in precedenza, alla filiera catodica si aggiunge un piano suddiviso in "pads", ossia una suddivisione in settori che consente di localizzare, oltre al filo che ha ricevuto il segnale, anche la posizione lungo esso in cui è stato ricevuto. Ciò serve a determinare entrambe le coordinate del punto sul piano dell'area di letture, che è quello indicato finora con xy. Il metodo usato è quello del baricentro, in cui si calcola la posizione con una media sui pad attivati, pesata sull'intensità del segnale ricevuto. In questo modo, se il filo colpito rappresenta il modulo della posizione registrata, il pad restituisce l'anomalia. In realtà, quindi, quando una particella lascia un segnale, non genera un punto, se cosí si conviene di chiamare una configurazione pad-row definita, ma piuttosto un cluster di punti vicini, dai quali se ne deduce uno preciso con il metodo del baricentro. Naturalmente, possono anche avvenire sovrapposizioni di cluster, e a questo punto entra in gioco il potere risolutivo della TPC che deve essere in grado di risalire meglio possibile a calcolare i centroidi dei due cluster sovrapposti, riuscendo cosí a separarli opportunamente. Naturalmente, la dimensione prevista di questi cluster è un parametro importante, in quanto determina quanto possono essere vicini due punti per essere distinguibili dalla TPC. Se, infatti, i cluster hanno una determinata dimensione media, possono avvicinarsi fino ad un certo valore massimo per essere distinguibili, e di conseguenza quella è la massima risoluzione della TPC. In funzione di questo discorso, è stata stabilita la dimensione dell'area di un pad-row, in modo da ottenere la dimensione minima di cluster, che ha comportato una diversa area da assegnare a ciascun elemento sensibile, andando dall'interno all'esterno. Il motivo di questo è, naturalmente, il fatto che la densità di tracce è più elevata nella regione interna, mentre all'esterno diventa più facile separare i punti e non c'è bisogno, quindi, di un potere risolutivo altrettanto spinto. I pad interni, quindi, sono rettangoli di 4×7.5 mm^2, mentre quelli esterni sono di 6×10 e 6×15 mm^2, per un totale di 570312 pads di lettura.

In base alla strutturazione della TPC proposta fino a questo momento, che è quella usata per le simulazioni nelle quali si è utilizzata la rete neurale, la TPC contiene in totale 80 pad-row, e quindi questo è anche il numero massimo di punti che una traccia può lasciare nel rivelatore. Di questi pad-row, i primi 30 stanno nel primo settore, e i restanti 50 nel secondo, in modo che, essendo il distanziamento uniforme in ogni settore, si ottengono le seguenti valutazioni per le distanze tra elementi contigui:

- nella regione interna, i valori di raggio vettore che possono essere rivelati vanno tra $\sim$84.1 e $\sim$132.1 cm, per cui la distanza media tra due fili è di $\sim$1.6 cm;

- nella regione esterna, si va da ∼134.6 a ∼246.6 cm, per cui si ottiene un distanziamento medio di ∼2.2 cm
- tra le due camere si trova, come si è già detto, uno spessore morto di 2.5 cm;

Queste misure sono le differenze medie che si prevedono, quindi, tra i moduli dei punti di una traccia rivelata dalla TPC di ALICE. A ciò si aggiunge il fatto che, in genere, potrebbe darsi il caso in cui non tutti i pad-row forniscono un segnale al di sopra della soglia elettronica, quindi si generano ancor meno punti. Ciò potrebbe, nell'ottica dei ragionamenti che saranno proposti per l'implementazione dell'algoritmo, rendere impossibile la ricostruzione di detta traccia.

Dopo aver illustrato quanto era necessario tener presente circa il funzionamento e la risoluzione dei dati prodotti dalla TPC in uso, è il momento di proporre il modello di rete neurale dal quale si è preso spunto per determinare l'algoritmo di ricostruzione.

5.2 Modello di Denby-Peterson

5.2.1 Considerazioni preliminari

I dati che vengono estratti dalla TPC sono essenzialmente le coordinate dei punti corrispondenti all'intersezione delle tracce di ogni particella con i diversi pad-row della TPC. Il problema da risolvere è, quindi, quello di stabilire quali punti appartengano a ciascuna traccia, per risalire poi all'impulso della particella che l'ha prodotta. L'ottimizzazione, quindi, consiste nello stabilire, tra le diverse sequenze che è possibile costruire con i punti ottenuti, quali siano quelle corrette. Naturalmente, per stabilire quando una traccia "va bene", bisogna avere un'idea della sua forma, e anche di quanti punti deve contenere. Per rispondere a queste due domande, va tenuto in considerazione lo spazio in cui le tracce si muovono e la fisica del loro moto. Quanto al numero di punti, si è detto che la TPC ha la possibilità di restituire un massimo di 80 punti per ogni traccia. Secondo le convenzioni standard adottate nelle descrizioni sperimentali usate [TDR-TPC, ALICE-FTF], una traccia "buona" deve contenere almeno il 40% del massimo numero di punti possibile, ossia 32 nel caso in questione.

Per stabilire la forma della traccia, si deve considerare, intanto, che una particella è "visibile" al rivelatore solo se possiede una carica. D'altro canto, si è detto che è mantenuto, nello spazio in cui le particelle si muovono, un campo magnetico $\mathbf{B}$, uniforme e giacente sulla direzione dei fasci collidenti, che sarà l'asse z in tutto il resto del discorso. In queste condizioni, se le particelle hanno notevole energia, il che permette di considerarle come entità classiche, abbastanza ben localizzate nello spazio, è possibile determinarne le equazioni del moto, conoscendone la carica q, con la formula della forza di Lorentz:

$$\mathbf{F} = q\mathbf{v} \times \mathbf{B} = \frac{d\mathbf{p}}{dt} \tag{5.1}$$

È noto come, in questo caso, la traiettoria della particella sia un'elica che avanza lungo l'asse z, mentre descrive una circonferenza nel piano xy. Le equazioni di quest'elica possono essere espresse ponendo x e y in funzione di z:

$$\begin{cases} x = x_0 + R\cos(\omega t + \phi) \\ y = y_0 + R\sin(\omega t + \phi) \end{cases} \tag{5.2}$$

Le costanti che compaiono nella (5.2) determinano le caratteristiche geometriche dell'elica nonché il suo punto di partenza che, dappertutto nel seguito è considerato essere sempre l'origine, essendo esso il punto in cui si innesca la reazione che produce le particelle rivelate.

Dal punto di vista dei dati sperimentali, naturalmente, una "traccia" non viene individuata da parametri geometrici che la determinano perfettamente. Ciò che si riceve dal rivelatore, infatti, è una linea poligonale che congiunge, nel dovuto ordine, i punti correttamente associati. Scopo della rete neurale sarà quindi quello di costruire innanzitutto le giuste poligonali; in seguito verrà fatto un best-fit, dal quale si ricostruiranno i parametri geometrici dell'elica e da essi i valori fisicamente significativi della traccia, quale ad esempio l'impulso della particella. A questo punto bisogna fare le dovute considerazioni che consentano di "mappare il problema", come si suole indicare il processo attraverso cui la questione fisica viene tradotta con la rete neurale.

Una prima osservazione è di carattere geometrico. Nel caso della TPC, che è il più efficace dei rivelatori atti alla ricostruzione delle tracce in ALICE, visto il gran numero di punti che restituisce, si è visto che due pad-row distano fra loro, in media, circa 2 centimetri. È anche chiaro che, dati gli impulsi elevati previsti per la maggior parte delle particelle rivelate di interesse sperimentale, le circonferenze che esse descrivono sul piano xy avranno un raggio notevole, paragonabile con quello dalla TPC stessa. Ora, una corda di 4 cm confrontata con un raggio di circa 300 cm sottende un angolo che ha un coseno che può essere approssimato ad 1 fino all'ordine della quinta cifra decimale. In altre parole, tale angolo è praticamente trascurabile, e questo permette di pensare che la corda e il corrispondente arco si confondano. Da questa deduzione, infine, si può concludere che, prendendo i due segmenti che congiungono tre punti consecutivi, è possibile considerarli come due parti di un unico segmento rettilineo, a parte approssimazioni di ordine elevato. Ciò giustifica un criterio che ritenga appartenenti ad una stessa traccia due segmenti allineati lungo una direzione radiale a partire dall'origine, che congiungono tre punti, condividendone uno.

Naturalmente, questo ragionamento diventa tanto meno consistente, quanto più sono lunghi i segmenti e quanto minore è l'impulso della particella, in quanto la curvatura della traccia influisce in modo più consistente. È anche vero che appare abbastanza difficile che nel rivelatore una traccia contenga punti consecutivi troppo distanti (anche se è probabile che non vengano colpiti tutti i pad-row; infatti, è difficile che due punti consecutivi differiscano di più di 5 unità). Tra l'altro, sarebbe sbagliato, in questo caso, tenere in considerazione non solo le coppie di segmenti allineati, ma perfino i segmenti stessi, in quanto è molto poco probabile che una particella percorra un tratto rettilineo da un punto all'altro, quando essi sono abbastanza lontani. Per questo motivo non ha senso considerare tutti i possibili accoppiamenti di punti, ma solo quelli tra punti abbastanza vicini. Inoltre, è chiaro da questo ragionamento che, quando due segmenti di diversa lunghezza si adattano bene ad un terzo, generalmente è preferibile associarvi il più breve, che sicuramente è un'approssimazione migliore del tratto che la particella potrebbe aver percorso tra i suoi estremi.

Queste due considerazioni geometriche servono a giustificare il modello di mappatura del problema del tracking che si intende seguire.

5.2.2 Descrizione del modello

Il modello studiato, del quale si intende proporre l'applicazione, è stato proposto da Denby e Peterson [Pet89, Den88] e si basa sullo schema topologico delle reti di Hopfield. In passato, tale approccio è stato riproposto pure da Stimpfl-Abele e Garrido in un'applicazione

relativa alla TPC del rivelatore ALEPH [Sti91], la quale però, a differenza di quella di ALICE, prevedeva una molteplicità di solo 100 tracce circa. Come ogni modello di rete neurale preposta all'ottimizzazione, quello di Denby e Peterson (che si definirà in sintesi DPM, da "Denby-Peterson Model"), va discusso in alcune fasi, che sono elencate di seguito a scopo di chiarezza e schematicità:

1. stabilire quale concetto fisico va associato ai neuroni della rete;

2. determinare le relazioni topologiche tra i neuroni;

3. stabilire i pesi sinaptici tra i neuroni;

4. impostare il criterio di aggiornamento.

Avendo affermato che la traccia sperimentale è una poligonale, è chiaro come si possa mappare il problema in una rete di Hopfield, rispondendo al quesito proposto dal punto 1. Visto che il problema è quello di trovare i lati delle poligonali corrette, i neuroni della rete saranno tutti i diversi lati plausibili fisicamente. Quindi ad ognuno sarà associata una coppia di punti sperimentali. Per questo motivo, immaginando di aver ordinato i punti etichettandoli un indice intero, qualsiasi grandezza relativa ad un neurone sarà rappresentata da una scrittura del tipo 's_{ij}', in cui i e j rappresentano gli estremi del segmento associato al neurone e s rappresenta la variabile fisica in questione (attivazione, lunghezza, ecc). In questo schema concettuale, allora, è chiaro che lo scopo dell'algoritmo neurale è quello di restituire uno stato stabile in cui solo i neuroni che rappresentano le sequenze di segmenti di traccia corretti siano accesi, e si spengano invece quelli associati a segmenti che congiungono punti di tracce diverse.

I pesi sinaptici verranno stabiliti in base a considerazioni geometriche, a partire dal significato stesso dei neuroni. Intanto, in analogia con la scrittura usata per le unità elementari, un peso sarà rappresentato da w_{ijkl} , in cui i e j sono gli indici degli estremi di una delle due unità e k ed l quelli dell'altra. Per quanto riguarda proprio il valore assegnato ai pesi sinaptici in questa rete neurale, il DBM, riallacciandosi alle considerazioni geometriche proposte in precedenza, trae le seguenti conclusioni:

1. due segmenti neurali appartenenti alla stessa traccia possono essere considerati, con un'ottima approssimazione parziale (cioè relativa alla regione di spazio in cui i segmenti si trovano), allineati;

2. ove c'è necessità di scegliere tra due segmenti neurali, per "seguire" la traccia a partire da un certo punto, il più corto è in genere il più plausibile;

3. la relazione tra due segmenti non aventi almeno un punto in comune non è semplice da ricavare, in quanto la loro distanza potrebbe rendere necessario considerare l'influenza della curvatura nel campo magnetico o anche fenomeni fisici avvenuti nel cammino della particella.

A queste tre considerazioni prettamente fisiche e geometriche, se ne aggiunge una quarta, esposta a parte perché richiede un ragionamento qualitativo opportuno. Tale considerazione nasce dalla necessità di "percorrere" la traccia ricostruita in un senso univoco. È importante che la rete abbia modo di stabilire quanto è plausibile che una certa sequenza di punti rappresenti una traccia non solo dalla forma, ma anche in base alla sequenza con cui

questi punti sono ordinati. Tale sequenza, infatti, non va determinata a posteriori, con criteri che prendono le mosse, magari, dalla distanza dei diversi punti rispetto all'origine: essa deve essere insita nella soluzione. Essendo infatti i punti disposti in una poligonale, il modo di seguire la traccia consiste essenzialmente nel passare da un punto all'altro seguendo il segmento che li congiunge nel giusto senso di percorrenza, e questo senso di percorrenza deve essere lo stesso che la particella, a suo tempo, ha seguito (o il suo inverso esatto). Come tale deve essere definito nella struttura della rete neurale e non dedotto dalla sua soluzione. Per questo motivo, diventa importante stabilire il verso di ogni segmento, che è correlato con la consequenzialità temporale che lega i suoi estremi, dipendente dall'ordine temporale fisicamente oggettivo secondo il quale un segmento viene percorso. Questo serve ad evitare che una traccia venga ricostruita in modo da rispettare solo la forma prevista, ma non la consequenzialità corretta tra i punti. Tale discorso serve a giustificare la scelta di associare, ad ogni segmento-neurone, un verso, in base al quale un estremo ne costituirà la "testa" e l'altro la "coda". Così , due segmenti potranno appartenere alla stessa traccia solo se, oltre ad avere un estremo in comune, risulteranno consecutivi in base alla sequenza stabilita "coda" $\rightarrow$ "testa".

In questo modo, inoltre, si possono stabilire in modo semplice le connessioni inibitorie tra i neuroni: due segmenti che hanno la "testa" o la "coda" in comune saranno in competizione, in quanto ogni punto può appartenere ad una traccia soltanto e quindi tra questi due, uno dovrà essere spento. Allora si può dire che due neuroni sono in relazione solo se hanno un punto in comune, ma la relazione sinaptica tra essi si può concretizzare in una "collaborazione" od in una "competizione", a seconda che essi, presi insieme, costituiscano una "sequenza" od un "incrocio". Naturalmente, infine, in questa rappresentazione, esprimere l'attivazione con s_{ij} implica che il neurone va percorso dal punto i al punto j, ma tutti i sensi di percorrenza possono essere invertiti temporalmente, in modo che la traccia venga percorsa, idealmente, al rovescio. Questo accorgimento, usato in molte procedure di tracciamento, è utile perché fa partire il processo di ricostruzione dalle regioni più esterne, in cui le tracce sono più diradate e vi è meno rischio di confusione, toccando le zone interne in un secondo momento, in cui i molti punti già trovati favoriscono una scelta più precisa dove la densità di tracce è più elevata.

In base a tutte queste considerazioni, il DBM propone, nel caso di neuroni consecutivi, un peso sinaptico di questo genere:

$$W^E_{ijkl} = B\frac{\cos^m \theta_{ijl}}{r_{ij} + r_{jl}}\delta_{jk} \tag{5.3}$$

Il coseno è correlato con l'allineamento dei segmenti, infatti tende ad 1 per segmenti molto ben allineati, ed a 0 nel caso opposto. In più, nel caso di neuroni posti in direzioni opposte, tende a -1, per cui rende inibitorio il peso, il che è giusto, perché non è compatibile con la fisica del moto di una particella il fatto che essa possa "tornare indietro" ad un certo punto del cammino, se i campi elettrico e magnetico che la fanno muovere si mantengono stabili. La presenza di un esponente serve a selezionare con maggior forza gli allineamenti migliori. Il denominatore, invece, è tanto maggiore quanto più sono lunghi i segmenti, per cui il peso loro associato risulta minore, e così si indeboliscono le correlazioni tra segmenti lunghi. Il coefficiente B serve perché il coseno è sempre compreso tra -1 ed 1, per cui, volendo far sì che il peso sinaptico possa essere maggiore, occorre moltiplicare per un valore positivo. Naturalmente, si suppone di porre le lunghezze a denominatore in unità che permettano di esprimerle come valori maggiori di 1, il che è successo nell'analisi

effettuata. Infine, il simbolo δ di Kronecker serve ad evitare correlazioni tra neuroni che non siano consecutivi, nel senso espresso in precedenza.

In modo più semplice, la correlazione tra neuroni in competizione, può essere espressa da un valore negativo $-A$, sempre uguale. Infatti, se vi è la competizione, non ci sono altri parametri da valutare per stabilirne l'intensità, dato che essa deve spegnere uno dei due elementi. Naturalmente si spegnerà l'unità che ha meno "sostegno" dalle sue vicine consecutive, segno che ha una minore probabilità di essere quella corretta. In linea di principio, potrebbe essere usato anche qui un peso sinaptico più complesso [Den00]:

$$W^I_{ijkl} = -\frac{A}{r_{ij} + r_{kl}} \delta_{jl} \delta_{ik} \tag{5.4}$$

in modo da rendere l'inibizione meno forte per segmenti più lunghi, che debbono risentire meno di tutte le correzioni, essendo meno passibili di "interessamento" da parte della rete neurale proposta. I simboli δ di Kronecker, stavolta, fanno in modo da selezionare solo neuroni aventi in comune un estremo con lo stesso nome: due "code" o due "teste". Quanto al criterio di analisi ed aggiornamento, si utilizza quello della rete MFT, per cui le attivazioni neurali sono valori reali compresi tra 0 ed 1. In base a questo criterio, l'aggiornamento dell'attivazione V_{ij} dell'i-esimo neurone, viene calcolata come segue:

$$V_{ij} = \left[1 + \exp\left(- \frac{\sum_k W^E_{ijjk} V_{jk} + \sum_k W^E_{kiij} V_{ki} - \sum_{k \neq j} W^I_{ijik} V_{ik} - \sum_{k \neq i} W^I_{ijkj} V_{kj}}{T} \right) \right]^{-1} \tag{5.5}$$

in cui la prima somma cerca tutti i segmenti che hanno la "coda" uguale alla "testa" di V_{ij} e la seconda cerca tutti quelli che hanno la "testa" sulla sua "coda". Le altre due sommatorie, invece, contano tutti gli elementi che hanno "testa" uguale e "coda" diversa da V_{ij} o viceversa, sottraendone le attivazioni, moltiplicate per il peso sinaptico di cui si è detto in precedenza. Si può riguardare questo contributo come l'espressione matematica di un problema di minimo vincolato in cui si impone che tra tutti i segmenti che iniziano nel punto i e terminano nel punto j deve essercene solo uno attivo.

In base a questo schema concettuale, allora, ciò che la rete dovrebbe restituire, nel migliore dei casi, sarebbe una mappa in cui i neuroni accesi siano disposti in modo da costituire un certo numero di linee poligonali aperte, dirette dalla zona centrale del rivelatore verso l'esterno, ed aventi una forma che richiami l'andamento di un'elica. Nel caso peggiore, si potrebbe pensare che comunque, in qualche modo si crei una configurazione simile ad un'immagine visiva "sfuocata", in cui, anche se non si distinguono correttamente i bordi, si può determinarli in base ad un'analisi della figura. Infatti si può pensare che, in un processo realistico con attivazioni reali, la rete si possa stabilizzare in base ai criteri sfruttati per valutare lo stato del sistema, quando alcune biforcazioni in entrata od in uscita non sono ancora state risolte, per cui non si ottengono proprio delle poligonali assolutamente prive di biforcazioni o di punti di confusione. In questo caso, però, il fatto che si siano usate attivazioni reali può risolvere la questione, se si sceglie di considerare "buono", tra più elementi in competizione, quello con l'attivazione maggiore, oppure quello meglio allineato, il che rientra, tra l'altro, nella logica di una "confidenza" nel criterio di aggiornamento della RNA.

Lo schema sinaptico proposto finora, però, non prende in considerazione i fenomeni che si verificano normalmente lungo il cammino di una particella: il decadimento e lo scattering multiplo. Il primo fenomeno comporta un processo che, in termini di dati ottenuti, produce

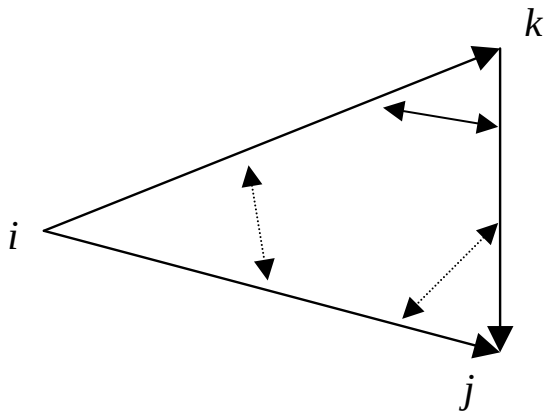

Figura 5.5: Le lettere i, j, k rappresentano i punti, e i segmenti orientati che le congiungono sono i neuroni. Tenendo conto del verso di percorrenza e di quanto esposto nel testo, le linee con due frecce rappresentano il tipo di peso sinaptico tra ogni coppia. La linea tratteggiata indica la connessione inibitoria, mentre quella continua rappresenta una connessione eccitativa.

la sparizione di una traccia e la comparsa, nello stesso punto, di altre due o più, con parametri specifici molto diversi, in quanto, si è visto sopra, ciò che determina passo, centro e raggio di curvatura della traccia sono la carica e l'impulso trasverso della particella. Ad aumentare la confusione contribuisce la possibilità che il decadimento produca una particella neutra. Non essendo essa visibile al rivelatore, il risultato che un tale decadimento produce è un brusco cambiamento di forma della traccia e la comparsa di una nuova curva più avanti, nel punto in cui la particella neutra prodotta, se non è stabile, decade in un'altra o più particelle cariche.

Dato che "seguire" la traccia attraverso questi cambiamenti non è facile si può dividere il problema in più elementi distinti, supponendo che la traccia "madre" si esaurisca nel punto in cui è avvenuto il decadimento, e le "figlie" inizino dallo stesso punto, o meglio dallo strato sensibile successivo, in cui esse lasciano segnali distinti. Tenendo conto del fatto che il punto di partenza è un parametro fondamentale da raccogliere, potrà essere poi ricostruito il decadimento avvenuto, in un'analisi fatta a posteriori, in cui si valutano le tracce fisiche, derivate dal best-fit dei dati analizzati con la rete neurale. Naturalmente, in questo modo, bisogna rivedere il presupposto secondo il quale le tracce provengono solo dall'origine, e ciò influisce essenzialmente sulla procedura di best-fit.

Quanto allo scattering multiplo, esso è un fenomeno che avviene in qualunque materiale in cui la particella passa ed è correlato con la perdita di energia. Tale processo, essendo totalmente casuale e tanto più debole quanto meno è denso il mezzo in cui la particella si muove, può, in prima approssimazione, essere semplicemente incluso nelle indeterminazioni sulla forma della curva, ponendo che uno di questi fenomeni non comporti deviazioni estremamente notevoli dalla traccia di partenza, salvo casi estremamente rari. In questi ultimi, la traccia è soggetta a brusche deviazioni, le quali comportano cambiamenti sostanziali nei suoi parametri di forma, dal momento che l'urto comporta variazioni tanto nella direzione e verso dell'impulso quanto nel suo modulo, tutte e tre determinanti per la forma fisica dell'elica descritta. Allora, nel caso di una traccia fortemente deviata da un fenomeno del genere, si deve necessariamente trattare il problema come se si avessero due tracce, riconducendo l'analisi allo stesso discorso fatto per il decadimento. Semmai, si

potrebbe distinguere questo caso dal precedente in base al fatto che, dalle eliche ricostruite, non si nota nessun fenomeno riconducibile a decadimenti scaturiti dal punto di deviazione, stabilendo così che il fenomeno è stato uno scattering multiplo molto intenso. Nel prosieguo della discussione non si farà ulteriore menzione del decadimento e dello scattering multiplo, che, pur importanti, vengono al momento trascurati. La loro inclusione è complessa, da un punto di vista matematico, ed esula da questo lavoro di tesi. Tuttavia, la loro assenza sarà discussa nell'analisi qualitativa e quantitativa dei risultati.

Una considerazione che, per quanto ovvia, va tenuta nel dovuto conto, è il fatto che, in ogni evento, i punti ottenuti sono diversi, per cui i neuroni vanno creati volta per volta, in una configurazione che dipende dal risultato ottenuto. Si capisce, quindi, come questo metodo sia utile esclusivamente per un'implementazione algoritmica adatta ad un'elaborazione al computer, piuttosto che alla costruzione effettiva di una rete neurale. Questa osservazione ne comporta un'altra direttamente collegata: ogni volta bisogna definire da capo i pesi e non è possibile avere esempi da usare per un eventuale addestramento, in quanto ogni evento sperimentale è un campione da analizzare *ex-novo*. Fare diverse prove serve solo a capire quali possano essere i valori migliori da dare ai parametri liberi che compaiono nelle (5.3) e (5.4), ossia A, B, m e T.

A questo punto, nota la rappresentazione dei neuroni, i loro pesi ed il criterio di aggiornamento, si ha tutto ciò che serve per l'implementazione del programma. Nelle figure 5.6 e 5.7 è proposta un'immagine esemplificativa dell'input ricevuto da una rete neurale di questo tipo, tradotto in un'immagine grafica bidimensionale, ed il corrispondente output.

5.2.3 Considerazioni utili all'implementazione

Innanzitutto, bisogna considerare che si ha a disposizione un computer per simulare una rete neurale, per cui la sua quantità di memoria RAM costituisce un limite superiore alla quantità di dati che si potranno conservare in essa per i calcoli. Nella fattispecie, per tutte le prove descritte in questa sede sono state eseguite su di un PC dotato di un processore Pentium II a 450 MHz con una RAM di 256 Mb. Questi parametri costituiscono limiti invalicabili alla quantità di dati memorizzabili in pratica ed alla rapidità di esecuzione dell'intero iter di calcolo. Avere a che fare con strutture che, per quanto potenti, hanno comunque dei limiti, che, in base ai tentativi fatti, le mettono alla prova in molti casi, rende necessaria l'ottimizzazione dell'occupazione di RAM e la scelta dei criteri migliori con cui eseguire i calcoli. Naturalmente, come è facile comprendere avendo un minimo di cognizioni in campo informatico, le strutture che verranno utilizzate per implementare i neuroni ed i punti e quant'altro con essi correlato, saranno array, peraltro abbastanza estesi. Il fatto che, in molti casi, le correlazioni tra elementi non riguardino ogni coppia di essi ma solo alcune coppie precise, rende necessario trovare il modo per evitare, ogni volta, di dover cercare tra molti valori inutili. Naturalmente, per operare le selezioni, servono operazioni aggiuntive che, dal canto loro, prolungano l'esecuzione, per cui bisogna trovare il giusto compromesso tra tempo di CPU necessario e quantità di RAM da impiegare.

Per questo motivo, si è cercato di introdurre diversi elementi che permettano di selezionare a monte i valori da cercare, evitando di scorrere tutto l'array dei neuroni o dei pesi. A questo scopo, anzi, onde evitare di depositare in memoria molte copie dello stesso valore, si è cercato di implementare piuttosto che ogni peso, la funzione che lo genera, allo scopo di minimizzare la RAM occupata, a scapito di una leggermente maggiore lentezza del processo. Tra l'altro, l'uso del linguaggio `C++`, che è quello in cui tutto il codice è stato scritto, permette di impostare le funzioni più usate in un modo che esse vengono compilate

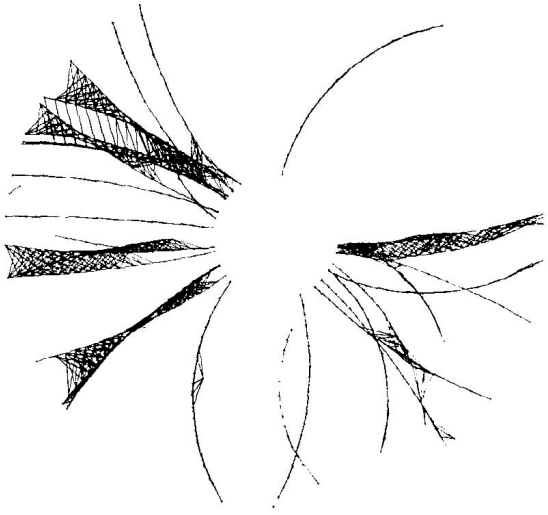

Figura 5.6: Immagine di una configurazione iniziale della rete di Denby e Peterson, con diversi reticoli in cui i punti vicini vengono congiunti da diversi segmenti.

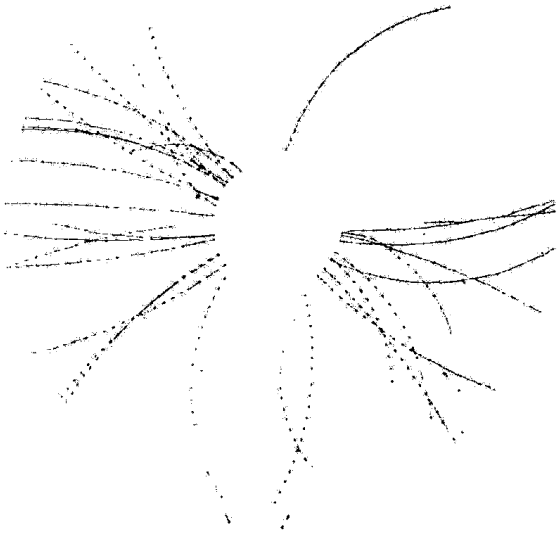

Figura 5.7: Immagine di una configurazione finale della stessa rete, in cui i segmenti "sbagliati" si spengono, e rimangono le tracce "ripulite".

direttamente in ogni punto in cui vengono richiamate, anziché definirle una volta e poi richiamarle ad ogni loro istanza. Ciò rende molto più rapida l'esecuzione delle stesse.

Un altro piccolo problema che è stato pure affrontato e risolto è stato quello dell'interfacciamento dei dati uscenti dal programma di simulazione di ALICE con il programma di analisi. Infatti, al momento, non è possibile disporre di dati sperimentali, essendo l'intero rivelatore ALICE ancora in fase di costruzione. Il modo in cui si sono ottenuti i dati è stato quindi quello di effettuare una simulazione al calcolatore di un evento Pb+Pb all'energia prevista di LHC all'interno di ALICE. A questo proposito è stato utilizzato il programma `AliROOT` [AliRoot], che è il package di simulazione e ricostruzione ufficiale della Collaborazione ALICE, basato su `ROOT` [ROOT] e `GEANT` [GEANT3]. Questo programma consta di tutte le procedure che, fornite dei dati relativi alla fisica del fenomeno da simulare, ossia impulso e natura delle particelle prodotte, loro quantità, possibili direzioni e fenomeni che possono avvenire (decadimenti, interazioni adroniche ed elettromagnetiche, produzione di particelle secondarie, ecc.), produce un risultato sperimentale simulato, restituendo un oggetto che contiene le tracce di tutte le particelle prodotte. Naturalmente, esso contiene le terne di coordinate di tutti i punti ottenuti dalla simulazione, per cui, ad analisi ultimata, si potrà valutare se essa è consistente o meno proprio in base alla corrispondenza del risultato ottenuto con quello iniziale. È questo il modo in cui una procedura di ricostruzione delle tracce viene testata prima di applicarla ai dati veri, perché si abbia una buona fiducia che l'interpretazione data di essi sia consistente, la qual cosa non potrebbe certo essere valutata se non da un confronto tra i risultati di metodi diversi. Una piccola nota a margine, comunque doverosa, è che, anche se potrebbe sembrare simile al processo di addestramento, questo confronto tra risultato ottenuto e risultato previsto non ha nulla a che fare con esso, in quanto, come dovrebbe essere chiaro da quanto detto finora, tale confronto non viene sfruttato per modificare i parametri di elaborazione della RNA, ma solo per stabilirne l'accuratezza e l'efficienza. Il procedimento di analisi è organizzato in una serie di passi:

1. lettura del file di testo contenente le informazioni sui punti sperimentali e registrazione in memoria di un array contenente tali informazioni;

2. creazione dei neuroni, con conseguente ricerca delle relazioni tra essi, e individuazione, per ogni punto registrato, di tutti i neuroni che lo hanno come "testa" e come "coda";

3. aggiornamento della rete con un criterio sequenziale, gestito in modo tale che ad ogni ciclo vengano aggiornati i neuroni più esterni prima di quelli più interni;

4. ricostruzione delle tracce finali trovate dalla rete neurale, e registrazione delle stesse su un file;

5. analisi dei punti trovati, e best-fit dei dati per l'ottenimento dei parametri fisici delle tracce, utili al confronto con i dati in partenza.

Le strutture di dati utilizzate sono due: una per il punto ed una per il neurone, che contengono, oltre ai dati di principale importanza, descritti in questa sede, alcuni riferimenti incrociati, utili a scorrere ciascun array senza perdite di tempo. Verranno esposte nel paragrafo successivo, in cui verrà descritto in dettaglio l'iter completo di elaborazione del programma.

Come si è detto, questo modello di rete neurale, come tutti quelli del tipo MFT, dipende, per la definizione dei pesi, da alcuni parametri liberi, che vanno fissati al valore opportuno:

- un valore per l'esponente m del coseno nella (5.3);
- un valore per il fattore inibitorio A nella (5.4);
- un valore per il fattore eccitativo generico B nella (5.3);
- un valore per la massima lunghezza plausibile di un segmento-neurone;
- la massima differenza di pad-row tra i punti collegabili da un neurone;
- il fattore di temperatura MFT;

Per quanto se ne sia discusso in precedenza, l'aggiunta di fattori di normalizzazione per le lunghezze, differenziati secondo il settore in cui il punto si trova, non è risultata di vitale importanza. Per questo essi non figurano tra i parametri e non vengono impostati in alcun modo. Invece, bisogna tenere in considerazione altri parametri, che servono a controllare l'aggiornamento e la ricostruzione delle tracce finali. Dal momento che si usano attivazioni reali, è poco verosimile sperare che ad un certo punto tutte le attivazioni restino esattamente uguali a se stesse, come avviene nella rete di Hopfield. Allora, come regola di stabilizzazione, si usa il seguente criterio:

$$\frac{1}{N}\sum_{i,j}\frac{\Delta V_{ij}}{V_{ij}} < \epsilon \tag{5.6}$$

ossia si controlla, dopo ogni ciclo, se la variazione relativa media delle unità è inferiore ad un valore specifico ϵ, e si ritiene stabile la rete quando ciò si verifica. Naturalmente, questo valore deve essere fornito, come gli altri, in una fase preliminare all'inizio dell'elaborazione neurale.

Un altro problema è quello della compatibilità tra segmenti. Il fatto è che i segmenti curvano solo sul piano xy, per cui è qui che vanno eseguite le analisi di allineamento utili alla ricostruzione delle tracce. Tuttavia, la scelta di costruire un segmento con due punti dipende anche da quanto tale segmento tende ad essere disposto nella direzione uscente dal vertice all'origine, e ciò viene controllato stabilendo una determinata differenza massima nelle coordinate angolari polari dei due estremi. Questi, quindi, sono altri valori che si aggiungono a quelli che bisogna definire.

La presenza di così tanti valori renderebbe un'interfaccia utente abbastanza complicata, a parte la necessità di ricordare ogni volta tutti i parametri immessi, per valutare, ad esempio, come cambia la bontà del risultato in casi diversi. Per ovviare a questo inconveniente, il modo di procedere seguito prevede la presenza di un file di testo ausiliario, in cui vengono specificati i valori di tutti questi parametri, preceduti da una parola chiave che consenta di riconoscere a quale elemento ogni valore va assegnato. In questo modo, tra l'altro, si può utilizzare lo stesso file di valori per più ricostruzioni e si rende più snella l'esecuzione. In appendice E viene proposto un brevissimo "manuale d'uso" dell'algoritmo, e vi compaiono anche le parole-chiave da usare per definire i diversi parametri.

Naturalmente, il programma di interpretazione produce un file di output, che contiene dati posti nello stesso formato di quelli in input, ma selezionati in ordine di traccia, nel senso che il file di output scrive i punti ordinati secondo le tracce riconosciute, di cui, però, elabora solo le coordinate. Tuttavia, i parametri aggiuntivi relativi all'impulso della traccia simulata, forniti nel vettore di input, sono necessari per valutare quanto bene la rete ha funzionato.

Un ultimo elemento da tener presente è che la rete neurale dipende da parametri geometrici, i quali debbono evitare di essere troppo "permissivi", altrimenti si impedisce alla rete di stabilizzarsi in uno stato plausibile (lunghezza ed accettanza in angolo innanzitutto, e anche minimo di stabilizzazione).

5.2.4 Struttura del programma

Adesso si passa ad esporre in termini di metalinguaggio le parti salienti del programma, esponendone in dettaglio il funzionamento, che sarà qui illustrato in modalità più discorsiva, mentre in appendice E si troveranno le istruzioni necessarie al suo utilizzo vero e proprio.

Una cosa da puntualizzare è la seguente: consultando la letteratura specifica esistente sull'argomento [Sti91], è stato possibile stabilire una procedura che serve a rendere più efficace il lavoro della rete neurale, eliminando molte correlazioni che, quando vengono considerate, conducono ad un alto rischio di interpretazioni sbagliate. In particolare, il problema sorge quando, essendo due tracce molto vicine, i loro punti vengono collegati da una fitta "ragnatela", che ha una grande probabilità di essere risolta alla fine in una poligonale che va a zig-zag fra le due, producendo quella che viene identificata come una *fake track*, il che non è assolutamente un risultato auspicabile. In figura 5.7 si notano alcuni casi del genere. L'algoritmo proposto nel riferimento citato, invece, opera una semplificazione radicale. Per ogni neurone, seleziona il miglior seguente, ossia quello che ha la coda sulla sua testa, ed il miglior precedente, ossia quello che ha la testa sulla sua coda, ove per "migliore" si intende quello cui è associato il peso sinaptico maggiore. Gli altri precedenti e seguenti vengono trascurati. In questo modo, un neurone riceve solo due input eccitativi, e molti inibitori, per cui esso tende a mantenersi attivo solo quando i due elementi associati contribuiscono con valori grandi, il che succede in particolare se sono ben allineati e brevi, cosa che generalmente è indice di una corretta associazione. Quando invece il miglior precedente ed il miglior seguente di un neurone formano angoli consistenti o sono lunghi, il loro contributo non riesce a sopraffare, ciclo per ciclo, i contributi inibitori, e di conseguenza, l'unità neurale si spegne. E' indicativo, tra l'altro, che alcune prove, fatte in una fase preliminare all'elaborazione del programma che sarà discusso di seguito, prove in cui non si era fatto alcun aggiornamento della rete, ma solo un concatenamento dei segmenti in base ad una regola di questo genere, conduceva molte tracce ad una identificazione notevolmente soddisfacente. Tuttavia, senza aggiornare la rete, gli errori casuali sono un rischio altissimo e ciò si è visto con simulazioni ad alta molteplicità, come sono quelle utilizzate per i test effettivi. Naturalmente, però, in questo tipo di configurazione, è necessario che il parametro moltiplicativo eccitativo sia maggiore di quello inibitorio, altrimenti tutti i neuroni tenderanno a spegnersi, rendendo impossibile alcuna interpretazione.

Altra aggiunta che viene fatta in [Sti91] all'algoritmo è quella di un *bias*, ossia un valore positivo non troppo grande, che viene aggiunto all'argomento della funzione logistica, all'atto di calcolare l'attivazione di ogni neurone. Naturalmente, questa aggiunta tende a veicolare tale unità verso l'attivazione, anche se è sbagliata. Quando, gradualmente, tale fattore di scala viene abbassato, rimarranno accesi solo i neuroni sostenuti da un contributo eccitativo veramente consistente, il che favorisce la selezione finale cui la rete intende giungere. Puntualizzate queste due premesse, è il momento di passare alla descrizione del programma.

Il primo passo è la definizione delle strutture di dati utilizzate per memorizzare i valori relativi a ciascun punto ed a ciascun neurone:

```
STRUCTURE Point {
```

```
    REAL x, y, z, phi, theta;
    REAL px, py, pz, pt;
    INT pad, n_in, n_out, in[], out[];
    INT ID1, ID2, ID3;
}
```

```
STRUCTURE Neuron {
   INT head, tail;
   INT pre, seq;
   REAL len, act;
   REAL wpre, wseq;
}
```

Le due strutture dati sono state scritte in modo da correlarne i campi in sottoinsiemi che saranno descritti di seguito. Per comprendere appieno il significato di alcuni elementi, però, bisogna premettere che nel programma si crea una coppia di array globali: `Hit[]`, di tipo `Point`, che raccoglie tutti i punti sperimentali, e `Unit[]`, di tipo `Neuron`, in cui vengono conservati i dati relativi a tutti i neuroni creati. Vengono definiti anche due interi, `Hit_N` ed `Unit_N`, che contengono la dimensione dei due array di strutture definiti. Gli indici di array vengono sfruttati per scorrere in maniera rapida i due vettori creati. Noto che sia questo discorso, ecco la descrizione dei vari campi delle due strutture.

1. Per quanto concerne la struttura `Point`:
   (a) nella prima riga compaiono le coordinate cartesiane, cilindriche e sferiche del punto;
   (b) nella seconda riga, si hanno le tre componenti cartesiane del vettore impulso associato al punto, e l'impulso trasverso;
   (c) nella terza riga si trovano due array di interi che servono a contenere gli indici di tutti gli elementi `Neuron` creati che entrano (`in`) o escono (`out`) da ciascun punto, con due interi che contengono la dimensione (risp. `n_in` e `n_out`) di ognuno dei due array;
   (d) nella quarta riga si trovano tre interi che rappresentano l'identificativo di traccia associato dal simulatore ad ogni punto.
2. Per quanto riguarda la struttura `Neuron`:
   (a) la prima riga contiene gli indici di array dei due oggetti `Point` che fungono da testa e coda dell'unità;
   (b) la seconda riga contiene gli indici dei due neuroni che costituiscono il miglior precedente ed il miglior seguente;
   (c) la terza riga contiene gli unici due dati fisici di interesse per un neurone, ossia la sua lunghezza (`len`) e la sua attivazione (`act`);
   (d) la quarta riga contiene i pesi sinaptici di correlazione con i due migliori neuroni di cui sopra.

Oltre a queste strutture, vanno definiti alcuni valori che contengono i parametri di controllo. Per potere in seguito fare riferimento rapidamente a tali elementi, li si elencano di seguito con i nomi che si sono dati loro nel programma stesso:

`Pad`	*indicatore della massima differenza in pad-row consentita*
`Phi, Theta`	*massime differenze delle omonime coordinate polari tra estremi di un segmento*
`Inh, Exc`	*valori moltiplicativi dei pesi sinaptici, rispettivamente inibitorio ed eccitativo*
`Pow`	*esponente del coseno nel peso eccitativo*
`Bias`	*fattore di bias per stimolare le attivazioni iniziali*
`Temp, Var`	*fattori di controllo della temperatura e della massima variazione consentita in un ciclo di aggiornamento*

È previsto l'uso di alcune funzioni, che servono a compiere alcuni calcoli matematici semplici, come quello della distanza fra due punti, oppure il peso sinaptico eccitativo. Non è il caso di esporre per esteso il listato di esse, dal momento che consiste in semplici calcoli diretti, che una minima esperienza di programmazione consente di implementare con notevole facilità. L'unica cosa da dire è che per il calcolo del coseno, si è sfruttato il prodotto scalare in tre dimensioni dei vettori rappresentati dai segmenti, con il verso definito in modo da andare dalla coda alla testa.

Onde evitare confusione inutile, anziché presentare schemi implementativi dello stesso genere di quelli proposti nei capitoli precedenti, data la maggiore complicatezza di quelli sviluppati nel programma realizzato, si è ritenuto più opportuno descriverne l'azione mediante schemi algoritmici (diagrammi di flusso) avulsi da qualsiasi paradigma implementativo.

Anzitutto, il programma, per esigenze di rapidità e snellezza della struttura, richiede il nome di un unico file di testo, in cui vengono specificati i valori per tutte le variabili presentate nell'elenco precedente, nonché i nomi dei file da cui prendere i punti e in cui salvare i risultati. In questo modo è possibile, tra l'altro, tenere traccia dei parametri impostati, in modo da poter valutare le scelte migliori per un buon risultato. La discussione inizia, quindi, presentando la routine di lettura dei parametri, in cui il nome del file da leggere compare in argomento: la figura 5.8 la illustra con un algoritmo, che si è ritenuta una maniera più immediata ed efficace per comprenderne il funzionamento.

La subroutine in figura 5.8 legge il file seguendo un criterio che dipende dalle impostazioni di cui maggior dettaglio si trova in appendice E. Si prevede che detto file sia costruito con una serie di righe di testo, in cui compare una *keyword*, seguita da un valore numerico (intero o reale) oppure un nome. Secondo la keyword letta, la subroutine assegna il valore successivo ad una delle variabili, oppure lo interpreta come il nome di uno dei file di dati che è necessario specificare. Tuttavia, onde evitare i problemi che nascono da parametri non corretti (che comportano la possibilità di divisioni per zero, radici di numeri negativi e altre situazioni matematicamente impossibili), tutti vengono inizialmente impostati a 0, in quanto è impossibile che abbiano valore minore o uguale a questo. Se, dopo aver letto tutti i valori, la subroutine ne trova uno minore o uguale a zero (il che può significare che è stato inserito in modo scorretto o manca nel file di impostazioni), essa si conclude restituendo un segnale di errore, al che il programma stesso termina senza operare alcuna analisi. Naturalmente, se tutto funziona correttamente, la subroutine lo segnala in modo da poter proseguire il lavoro. Si noti che anche il parametro inibitorio è positivo, il che è sempre possibile se si conviene di moltiplicarlo dopo averlo cambiato di segno.

Non è il caso di soffermarsi sulla routine che legge i dati dal file di punti, che è triviale nella sua implementazione. Si pensi semplicemente che essa apre un file di testo del tipo

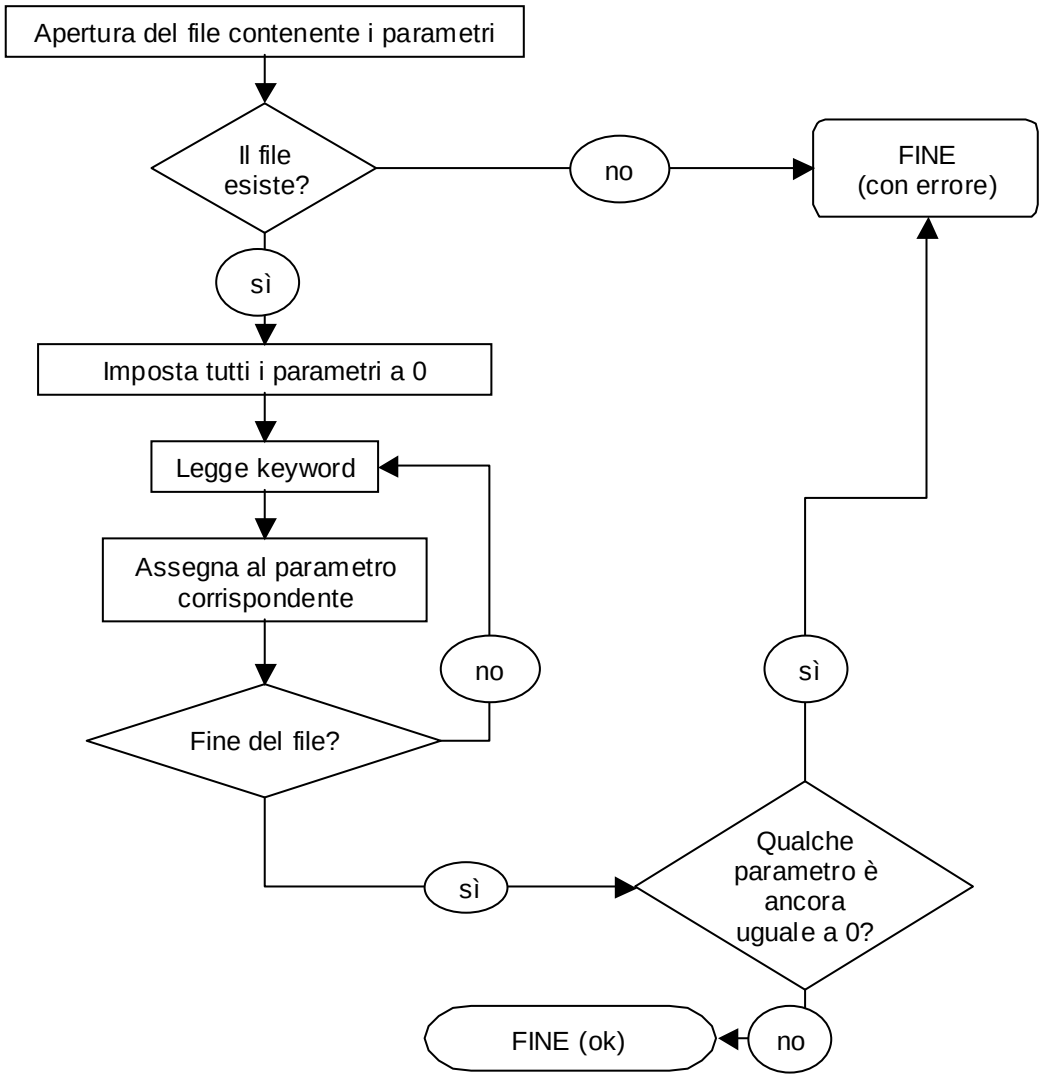

Figura 5.8: Diagramma di flusso della routine di lettura dei parametri.

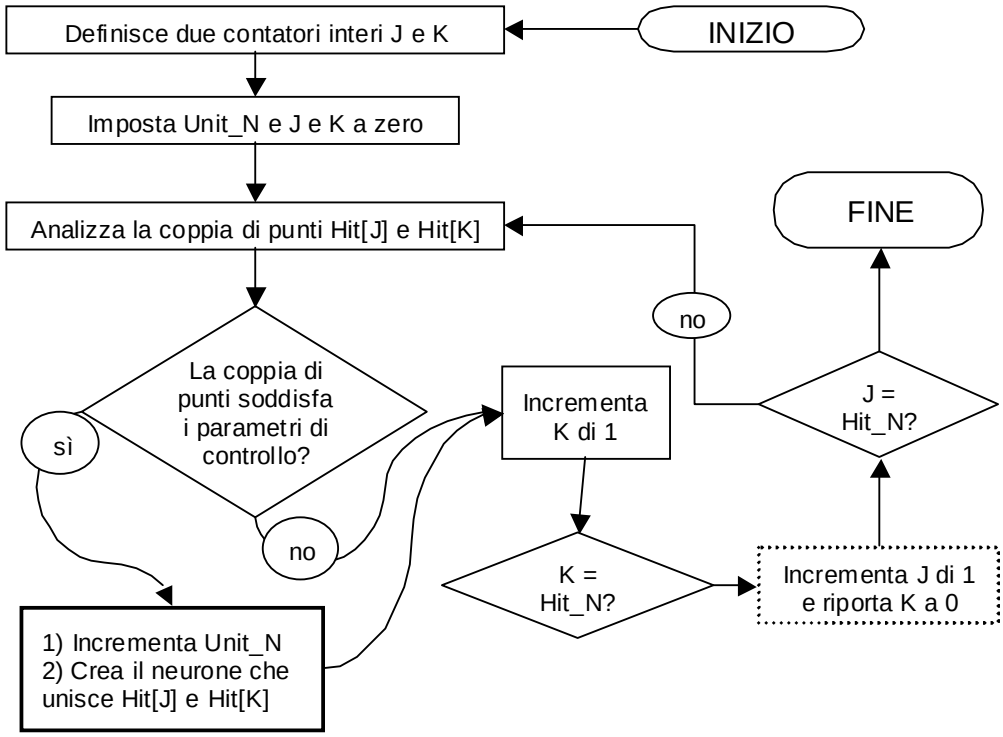

Figura 5.9: Diagramma di flusso della routine di creazione dei neuroni.

descritto in appendice E come file `.txt`, e lo legge per caricare nell'array `Hit` tutti i punti che trova, riempiendo tutti i campi della struttura associata.

Una volta riempito l'array di punti, prima di utilizzarlo per la costruzione dei neuroni, si rende necessario ordinarlo dal più distante al più vicino all'origine, in modo che la creazione dei neuroni, realizzata scorrendo questo array, dia indici minori ai neuroni relativi a hit più distanti dall'origine. Ciò serve per operare una procedura di aggiornamento che lavori *prima* sui neuroni esterni e *poi* su quelli interni. Eseguito l'ordinamento, l'array `Hit` diviene il riferimento essenziale per la creazione dei neuroni. Entra in gioco, infatti, la subroutine preposta alla creazione di questi ultimi, che analizza ogni coppia di punti e valuta se essa si trova entro tutte le limitazioni imposte dalle variabili di controllo definite. In questa fase, è già stato fissato il valore di `Hit_N` ma non ancora quello di `Unit_N`. Questo è bene saperlo perché l'impossibilità di conoscere tali valori prima dell'inizio comporta la necessità di impostare questi due array nel modo che, in informatica, viene definito con *allocazione dinamica della memoria*, ossia un array la cui dimensione viene stabilita durante l'esecuzione. In tutti i moderni linguaggi di programmazione si prevede di poter gestire una situazione del genere.

La figura 5.9 mostra il diagramma di flusso che illustra il funzionamento della subroutine di creazione.

Per non appesantire la figura, i diversi controlli che vanno eseguiti sono stati riassunti in un unico blocco, ma comunque la struttura del diagramma non è troppo lineare, e ciò è dovuto al fatto che fa due visite nidificate sull'array `Hit`, per confrontare ogni coppia di punti. L'accorgimento, espresso nel blocco col bordo tratteggiato, di impostare il contatore K (interno) ad una unità in più del contatore J (esterno) ogni volta che viene incrementato quest'ultimo, fa sì da evitare di confrontare ogni coppia di punti più di una volta.

Il blocco dal bordo spesso, invece, rappresenta il fatto che l'intera subroutine va eseguita due volte: una volta per valutare quante coppie "buone" ci siano, e una seconda volta per creare i neuroni. Il motivo è la necessità di avere il numero di neuroni per dimensionare l'array `Unit` prima di poterlo riempire, in modo da evitare carichi di memoria eccessivi (come si potrebbe avere dandogli una dimensione preliminare molto grande) o errori di sistema (come succederebbe se non lo si dimensionasse affatto). Si sarebbe potuto procedere con *liste concatenate*, una struttura complessa di cui si può trovare un accenno in [Free91], però si è notato, in un tentativo di implementazione, il grande rischio di errori concettuali dovuti ad una non professionale conoscenza del linguaggio di programmazione usato, nonché l'eccessivo carico di memoria ed il notevole rallentamento delle operazioni, rispetto al modello usato alla fine. Un ultimo appunto, che non è evidente dalla figura, è il seguente: dal momento che, come si è detto, è possibile percorrere una traccia al rovescio, ed inoltre, nella regione più lontana dall'origine le tracce sono più diradate, è utile orientare tutti i neuroni dall'esterno verso l'interno in modo che ogni neurone abbia verso entrante rispetto all'origine nel piano xy, per cui la sua testa è il punto più vicino ad essa. Ciò che si fa è praticamente un'inversione del tempo fisico di percorrenza della traccia, rispetto alla quale tutte le grandezze scalari, tra le quali si può includere il modulo dell'impulso, sono invarianti (ovviamente ciò è vero solo nell'assunzione fatta di trascurare la perdita di energia e la diffusione multipla). Un ultimo passo della routine consiste nel trovare, per ogni punto, quali neuroni vi entrano e quali ne escono, in modo da snellire le procedure di aggiornamento di cui si serve il programma in seguito.

A questa fase segue quella della ricerca, per ogni unità, del miglior precedente e del miglior seguente. A questo provvede un algoritmo come quello illustrato in figura 5.10.

Naturalmente, questa subroutine illustra soltanto come viene scelto il miglior precedente. Sostituendo, nella figura, il verbo "entrare" con "uscire" e "coda" con "testa", si ottiene, in modo semplice, l'immagine della subroutine che trova il miglior seguente. Esiste, ma è banale e non vien illustrata anche per evitare eccessivi dilungamenti, una terza fase di controllo, in cui, onde evitare "vicoli ciechi" o biforcazioni, viene controllato, per ogni neurone, se il precedente del suo seguente è esso stesso, e lo stesso con il seguente del suo precedente. In questo modo, per ogni unità ne esiste solo una che l'ha come miglior seguente e solo una che l'ha come miglior precedente. Tutte le altre che condividono un seguente od un precedente, vengono impostate come se non ne avessero. Queste sono le unità che, con grande probabilità, sono sbagliate, e così si spengono entro un ciclo dall'inizio, come è preferibile che sia. Tra l'altro, se non si operasse questa selezione, le biforcazioni si manterrebbero, senza un oggettivo vantaggio operazionale.

La procedura di aggiornamento della rete, secondo il modello MFT, opera in modo sequenziale, per cui, selezionato un neurone, ne calcola l'input complessivo, e lo inserisce ad argomento della funzione logistica, mediante la quale calcola la nuova attivazione. Dal momento che ogni neurone ha due soli contributi eccitativi, il calcolo di essi è immediato, inserendo le attivazioni del miglior precedente e del miglior seguente moltiplicate per i corrispondenti pesi sinaptici, che tra l'altro vengono memorizzati espressamente nei campi `wpre` e `wseq` della struttura `Neuron`, come si è visto precedentemente. Per cercare i contributi inibitori all'attivazione di un'unità, basta considerare tutti i neuroni che escono dalla sua coda, e tutti quelli che entrano nella sua testa. Immediatamente si può vedere che questi, e solo questi, sono gli elementi che sono in competizione con l'unità in fase di aggiornamento. Si capisce allora il motivo per cui, di ogni punto, vengono, in fase di costruzione, cercati tutti i neuroni entranti ed uscenti. Scorrere l'array dei punti è certamente meno laborioso che non scorrere l'array dei neuroni (che saranno dell'ordine di

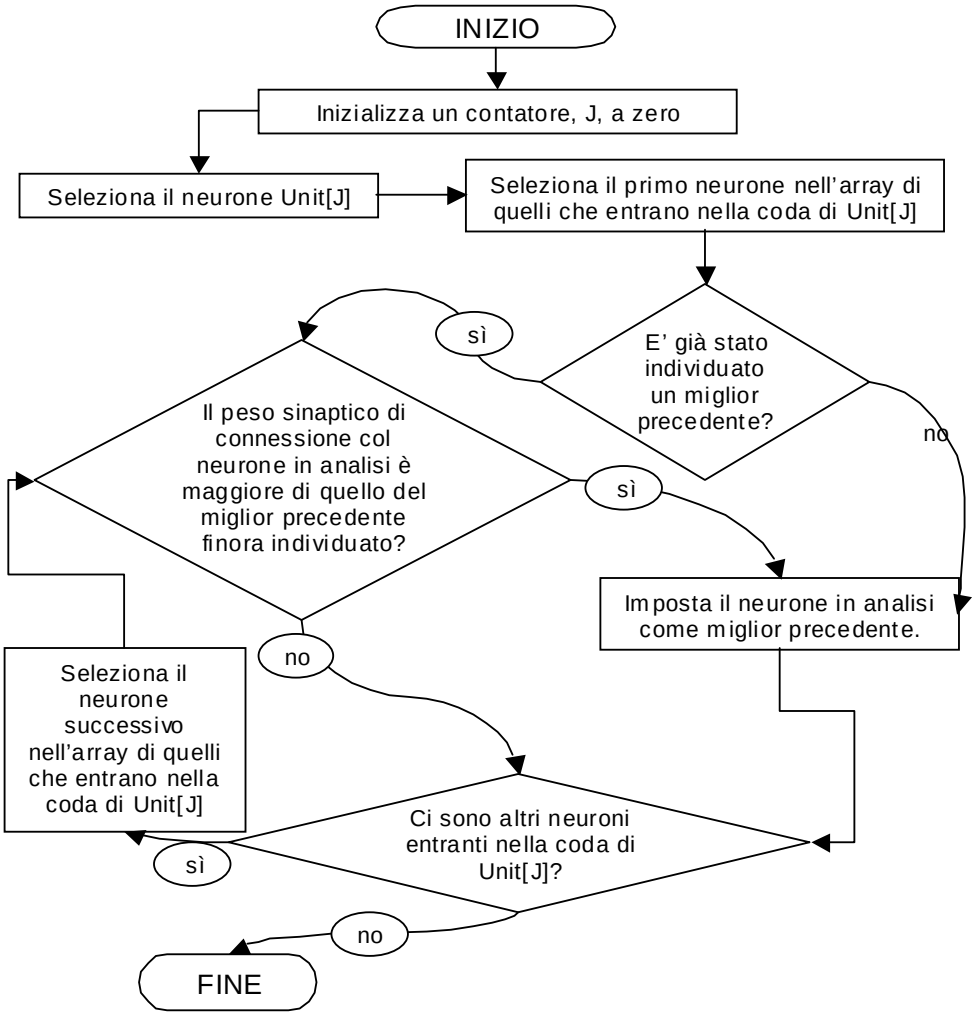

Figura 5.10: Diagramma di flusso della routine di selezione delle sequenze.

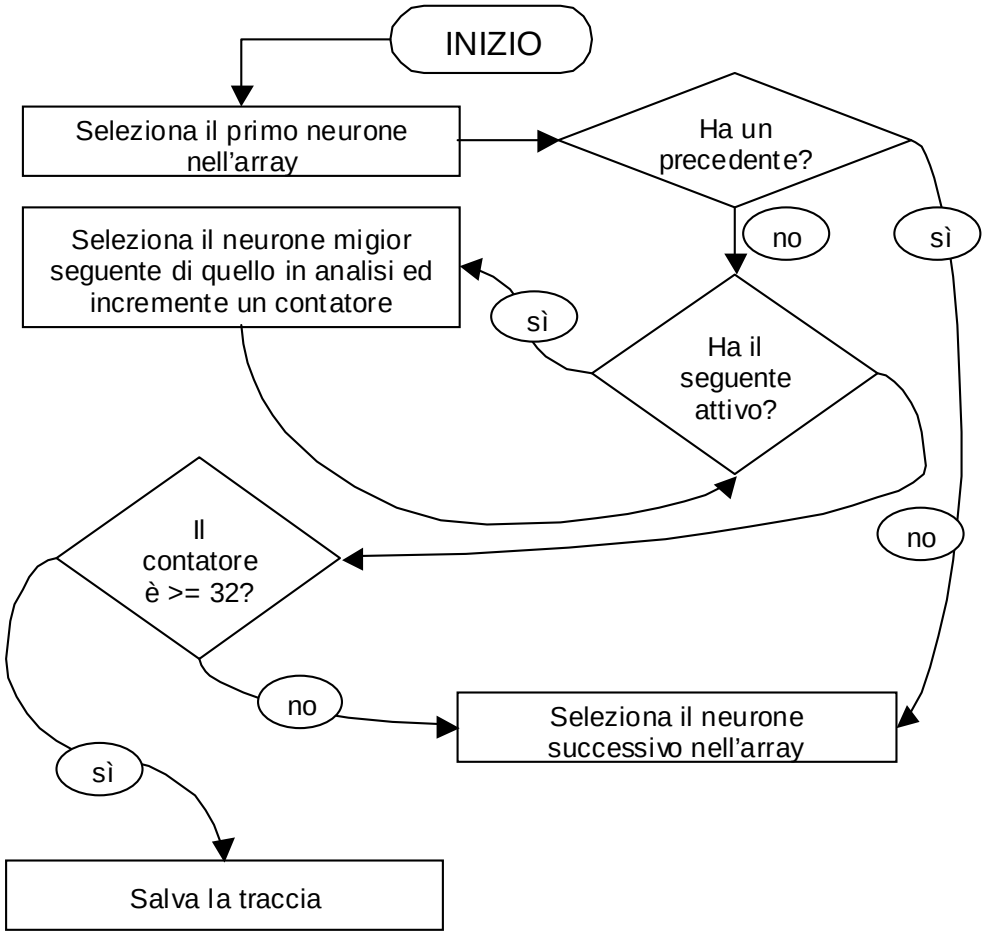

Figura 5.11: Diagramma di flusso della routine di ricostruzione delle tracce.

almeno 10 volte questi), e conoscere quali segmenti entrano ed escono da ciascun punto permette di evitare di scorrere ogni volta tutto l'array dei neuroni per vedere che tipo di contributo ciascuno di essi dà all'attivazione di uno in particolare. Un ultimo vantaggio è quello di evitare di testare la correlazione con tutti i neuroni che non condividono alcun punto con quello in fase di aggiornamento, riducendo di moltissimo i tempi di elaborazione.

Quando l'aggiornamento ha condotto alla stabilizzazione, bisogna alla fine ricostruire la traccia. Per eseguire questo lavoro evitando di "trovare" la stessa traccia più volte, si esegue un lavoro come quello espresso nell'algoritmo in figura 5.11.

Come è evidente, vengono analizzati solo i neuroni che non hanno un miglior precedente, quindi gli unici candidati ad esser l'inizio di una traccia. Ognuno di essi, se attivo, viene percorso in avanti, analizzando il suo seguente, poi il seguente del suo seguente, e così via. Nel frattempo viene incrementato un contatore che misura quanti elementi sono stati recuperati. Se il contatore raggiunge il valore 32, il minimo, in base alla convenzione seguita, per poter dire che la traccia è stata trovata, la traccia viene salvata, ripercorrendola nello stesso modo, e scrivendo in un file i punti trovati. A questo punto il lavoro della rete neurale è finito.

Quello che segue è l'interpretazione dei dati, ma per i dettagli di questo lavoro si rimanda all'appendice B per il best-fit e all'appendice E per i dettagli sul programma di valutazione. Qui, invece, si passa ad illustrare alcuni risultati conseguiti con quanto si è

riuscito a produrre con le simulazioni realizzate e con tutti i perfezionamenti operati sul software scritto.

5.3 Risultati

È il caso di premettere il tipo di risultati previsti, e di conseguenza il tipo di trattamento che viene eseguito sui risultati della rete neurale, per valutarne la risposta e poi la bontà di quest'ultima.

5.3.1 Considerazioni per l'analisi dei dati

La rete neurale, in ultima analisi, produce un file di dati in cui registra, per ogni traccia prodotta, una serie di dati per ogni punto. Per separare le diverse tracce, viene scritto ogni volta il numero di elementi di ciascuna, in modo che il programma di lettura successivo legga tale numero e sappia quanti punti deve registrare per ogni insieme.

La prima cosa da fare, per ogni traccia determinata dal programma neurale, è stabilire quante tracce ha trovato e se vanno bene o meno. Infatti, il programma di simulazione usato, prodotto nel framework `AliROOT`, in cui si sono realizzate tutte le simulazioni relative ai test di ALICE, assegna un'etichetta univoca ad ogni traccia, per cui è possibile stabilire se i punti delle tracce trovate dalla rete neurale corrispondono alla traccia originaria. A parte questo, è chiaro che una traccia deve essere definita nel modo migliore possibile, ma è anche vero che non sempre ad essa la rete assegna tutti i punti previsti, per cui potrebbero ottenersi tracce con meno degli 80 punti che teoricamente si attendono. A questo punto, per valutare quanto bene la rete neurale ha associato i punti, si applicano, oltre alla condizione di avere almeno 32 punti, ossia il 40% del totale, le seguenti convenzioni [TDR-TPC, ALICE-FTF], secondo cui:

- una traccia "buona" non deve contenere più del 10% di punti errati (ossia con etichetta di traccia originaria diversa da quella della maggioranza degli altri punti); quindi, in questo caso, sono ammissibili al massimo 8 errori;
- una traccia "buona" non deve avere più della metà di punti errati nei primi 10% più interni, ossia, in questo caso, tra i primi 8, almeno 4 debbono essere correttamente associati.

Sulla base di questi criteri valutativi, è possibile innanzitutto definire, quindi, quali tracce sono state trovate con successo e quante invece sono tracce sbagliate, quelle chiamate precedentemente *fake tracks*. Sulle tracce buone trovate, anzitutto, è importante stabilire una percentuale rispetto al totale di tracce buone prodotte, per cui stabilire che probabilità (efficienza) ci sia di ottenere buoni risultati.

In un secondo momento, è importante operare un best-fit delle tracce, che viene eseguito in base ai calcoli descritti più in dettaglio nell'appendice B. Trattandosi di traiettorie elicoidali, mediante una serie di trasformazioni, è possibile ricondurre la procedura di fit in quella di due fit di retta consecutivi, che conducono all'ottenimento dei seguenti parametri di interesse fisico:

1. il parametro d'impatto longitudinale, ossia la distanza della traccia dal piano xy quando la sua proiezione in questo piano passa per l'origine (questa definizione vale in base al *vertex constraint*, che è una condizione, imposta qui, secondo la quale la

proiezione della traccia sul piano xy passa per l'origine al tempo 0, ossia quando l'evento si verifica);

2. l'angolo azimutale ϕ dell'impulso vettoriale nel vertice;

3. l'angolo λ, complementare dell'angolo polare dello stesso vettore;

4. l'impulso trasverso, ossia la proiezione del vettore impulso sul piano xy.

Per comprendere il significato di questi parametri, si osservino le figure 5.12 e 5.13.

L'utilità dei parametri trovati è, naturalmente, quella di consentire la determinazione delle proprietà fisiche della particella che origina la traccia. Adesso è il caso di passare all'analisi vera e propria.

5.3.2 Risultati delle prove

Per soddisfare la necessità di accelerare i tempi di elaborazione, pur mantenendo la complessità del problema ai livelli corretti, il simulatore è regolabile in angolo di emissione. In pratica, anziché produrre una quantità di particelle tale da riempire tutto lo spazio di tracce, in quantità tale da raggiungere la molteplicità prevista per ALICE, si è limitata la regione di emissione alla zona compresa tra 30 e 70 gradi di anomalia, in modo da non costringere il computer utilizzato ad occupazioni di RAM troppo elevate. I file prodotti, sui quali è stata operata la prova, sono tre soltanto, in quanto non è necessario averne diversi esempi, se non per differenze di molteplicità:

- si è realizzato un file ad alta molteplicità, con 2700 particelle iniziali, tra cariche e neutre, che corrispondono alla massima molteplicità prevista nel caso di collisioni del tipo di quelle in programma per l'esperimento.

- un secondo file (molteplicità media) che ha una molteplicità pari alla metà della precedente;

- un terzo file (molteplicità bassa) che ha una molteplicità pari alla metà del precedente, quindi un quarto del massimo.

Si farà riferimento ai tre casi con le shortcut HIGH, MED e LOW nelle diverse tabelle o nei grafici, ove sarà necessario specificare. In questo modo, si può valutare quanto bene si comporti la rete, in base alle diverse possibili situazioni sperimentali, valutando se, ed eventualmente come, le prestazioni decrescano nel caso di molteplicità maggiori.

Quello che è stato variato maggiormente, sono le impostazioni dei parametri di controllo, che sono presenti nel programma, come descritto in precedenza, per cercare la configurazione a cui conseguisse il risultato migliore. Naturalmente, però, essendo questi parametri in numero di 5 (temperatura, inibizione, esponente del coseno, eccitazione, bias), trovare la configurazione assolutamente migliore non è operazione facile nè rapida, per cui ciò che si è fatto è stato un aggiustamento progressivo, regolato in funzione dei risultati, e cercando di mantenere i valori dei parametri entro numeri non troppo grandi. Non è certamente proponibile un'esplorazione dettagliata delle ∞^5 possibili combinazioni. Inoltre, il tempo di esecuzione del programma, come è prevedibile in seguito alle dimensioni degli array creati, rende anche dispendioso in termini di tempo eseguire molte prove, quindi anche questo è stato una limitazione alle risorse di cui si è potuto disporre. A questo

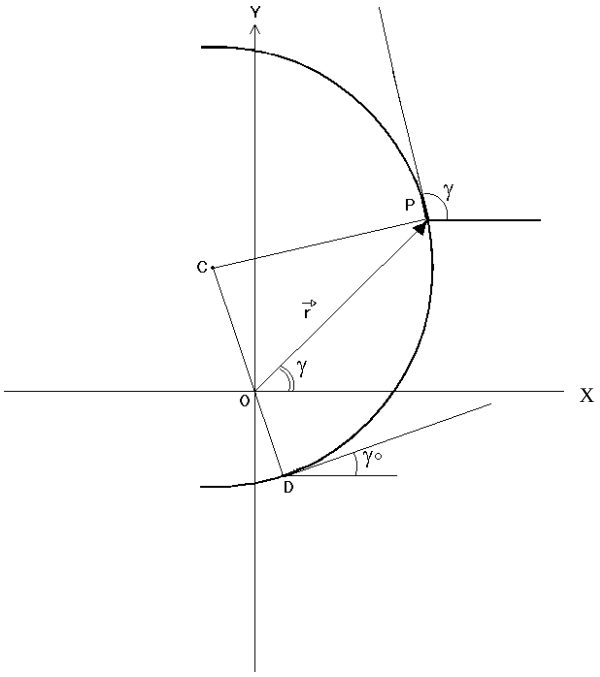

Figura 5.12: Proiezione di una traccia generica sul piano xy.

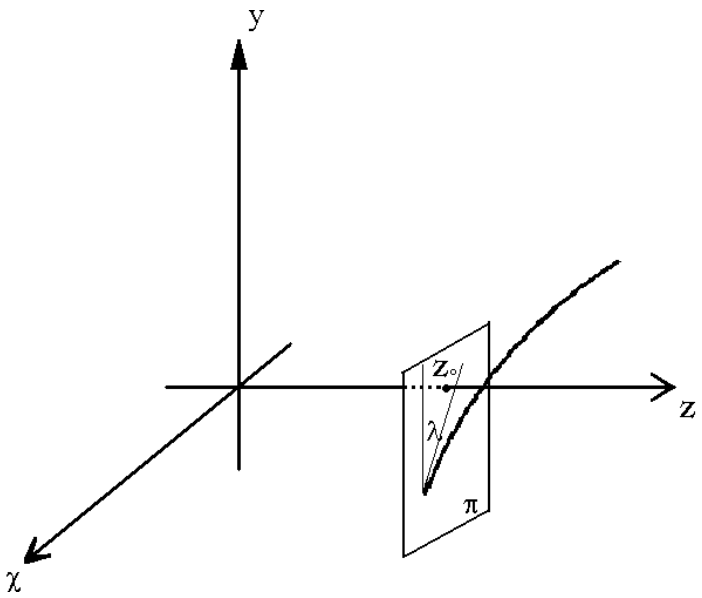

Figura 5.13: Diagramma assonometrico di una traccia in tra dimensioni.

File	Media neuroni	Media Tempo di CPU
LOW	~ 130000	50(1) s
MED	~ 480000	250(1) s
HIGH	~ 1460000	750(1) s

Tabella 5.1: Media di neuroni creati e di tempo di CPU impiegato per l'esecuzione del tracking neurale con le diverse molteplicità.

File	Kalman	FTF (10×10)	FTF (100×100)
LOW	17.6 s	2.2 s	0.7 s
MED	56.1 s	15.5 s	1.9 s
HIGH	189.6 s	162.3 s	5.1 s

Tabella 5.2: Media del tempo di CPU impiegato per l'esecuzione del tracking con le diverse molteplicità con il Kalman filter e con l'FTF. Nel caso dell'FTF, i due valori, 10x10 e 100x100, si riferiscono al numero di volumi in cui viene fatta la ricerca dei prolungamenti di traccia. Per maggiori dettagli si rimanda il lettore a [FTF].

proposito, anzi, è sembrato utile stimare il tempo di CPU impiegato dall'intero programma per operare l'intero processo, per avere una stima di applicabilità anche in questo senso.

Anzitutto, per una stima generica volta allo scopo di conoscere l'entità dello sforzo operato dal computer per eseguire il programma, si sono fatte diverse prove a caso, impostando diversi valori per i parametri di controllo, che hanno comportato, però, un impiego di tempo grosso modo indipendente dal numero di neuroni prodotti, nel caso in cui il file di input fosse lo stesso. La tabella 5.1 riporta il numero di neuroni creati ed il tempo di CPU impiegato per le diverse prove preliminari fatte.

Si riportano, per confronto, nella tabella 5.2 i risultati ottenuti con altri due metodi di ricostruzione, in fase di elaborazione e raffinamento più avanzate, che sono stati proposti per operare in ALICE: il Kalman filter [KALMAN] e l'FTF [FTF](Fast Track Finder), di cui si può trovare dettaglio e riferimenti in [ALICE-FTF]. Questi due metodi sono quelli rispetto ai quali va valutato il tipo di performance che la rete neurale riesce a dare.

Si può notare, nel tracking neurale, come la variazione del tempo di CPU sia molto sensibile passando da una molteplicità all'altra. Il motivo è che tale tempo dipende essenzialmente dalla quantità di elementi da aggiornare, ma soprattutto dalla quantità di competizioni da risolvere, sia per la scelta di precedenti e seguenti, sia per la ricerca di incroci per i contributi inibitori. Infatti, se la molteplicità aumenta, a parità di apertura angolare massima consentita, verranno costruiti molti più elementi, il che significa che, da ogni punto, entrano ed escono molti più elementi, e quindi, anche con l'accorgimento di visitare, ogni volta, solo gli elementi entranti od uscenti da un punto determinato, la ricerca è comunque più lunga. A questo si aggiunge la considerazione banale che il numero di punti è esso stesso maggiore. Da notare è che la scelta di ridurre a due o poco più i contributi eccitativi aiuta ad accelerare l'esecuzione, dato che il peso eccitativo va calcolato mediante una funzione, mentre quello inibitorio è determinato da una moltiplicazione, che è molto più rapida da eseguire per il processore di un computer. In questo modo si può contribuire a limitare il rallentamento delle operazioni.

La tabella 5.3 riporta, per le tre diverse molteplicità, l'efficienza media calcolata e la probabilità di fake tracks.

Per confronto, ecco, in tabella 5.4, le efficienze riportate per gli stessi altri due sistemi di ricostruzione di tracce introdotti in precedenza.

File	Eff. media	Prob. *fake*
LOW	$84 \pm 7\%$	$5 \pm 1\%$
MED	$72 \pm 4\%$	$13 \pm 2\%$
HIGH	$64 \pm 3\%$	$5 \pm 1\%$

Tabella 5.3: Efficienza media e probabilità di fake migliori ottenute nelle prove fatte con il programma di tracking neurale.

File	Kalman	FTF (10×10)	FTF (100×100)
LOW	99.3%	98.2%	98.2%
MED	96.0%	96.4%	97.5%
HIGH	93.5%	95.9%	95.3%

Tabella 5.4: Efficienza media di trackinf del Kalman filter e dell'FTF, riportata dal riferimento [ALICE-FTF]

Un'altra cosa da valutare è la corrispondenza tra i valori iniziali dei parametri fisici delle tracce ricostruite e quelli effettivamente calcolati in seguito al best-fit dei dati della rete neurale. Si è detto che le procedure di fit seguite non tengono conto, visto lo stadio sperimentale e progettuale del metodo proposto, di complicazioni molto specifiche quali potrebbero essere quelle relative allo scattering multiplo e, soprattutto, la perdita di energia che sono invece incluse in algoritmi in stato più avanzato di programmazione quale il Kalman filter. Per esempio, si presuppone che le tracce siano eliche perfette, mentre sarebbe più corretto immaginarle come spirali, dal momento che l'energia, e quindi l'impulso di ogni particella, decrescono in seguito all'attraversamento del volume della TPC, il che, tra l'altro, è proprio il fenomeno fisico che consente a questo rivelatore di restituire il segnale dell'avvenuto passaggio di ognuna.

Altra cosa da considerare è che il criterio secondo il quale viene ricostruita ogni traccia dalla rete neurale è basato su un'operazione concettuale ben differente da quella che ispira entrambi i metodi proposti a titolo di confronto. Infatti, entrambi questi ultimi operano direttamente sui punti, aggiungendone uno alla volta in una sequenza che viene man mano controllata per valutare tanto i parametri geometrici della traccia, quanto la direzione in cui cercare il prossimo punto. La rete neurale, invece, deve trovare la traccia tutta in una volta, riscontrando gli accordi giusti, direttamente tra tutti i punti. In questo senso non le è possibile operare aggiustamenti di sorta lungo il processo, ma l'unico modo per ottenere una buona performance è impostare correttamente i parametri di controllo. In questo senso, mettere l'algoritmo in condizioni di operare in modo efficiente è un lavoro abbastanza complesso.

Fatte queste precisazioni, ecco una tabella che riporta, per l'impulso trasverso e per gli angoli al vertice, i valori del centroide e della semi-larghezza σ provenienti da un best-fit gaussiano della distribuzione degli scarti rispetto ai valori iniziali (quelli della traccia simulata). Sono presentati, anche qui, i confronti con Kalman e FTF. I risultati sono proposti nelle tabelle 5.5, 5.6 e 5.7

I grafici presenti nelle pagine finali di questo capitolo mostrano, a scopo di maggiore immediatezza, gli istogrammi di efficienza del tracking neurale e, per confronto, quelli analoghi per il Kalman e per FTF, e a seguire, i risultati grafici dei best-fit sulle quattro grandezze fisiche misurate in ogni traccia, anche questi nel confronto tra rete neurale, Kalman ed FTF.

File	$\bar{p_t}$	σ_{p_t}
LOW	-1.24 ± 0.22	4.23 ± 0.17
MED	-0.23 ± 0.2	3.69 ± 0.15
HIGH	-0.94 ± 0.7	3.44 ± 0.15
File	λ	σ_λ
LOW	-0.48 ± 0.57	7.97 ± 0.87
MED	0.19 ± 0.55	6.93 ± 0.53
HIGH	0.79 ± 0.26	6.83 ± 0.28
File	$\bar{\phi}$	σ_ϕ
LOW	-0.24 ± 1.53	3.66 ± 17.62
MED	-0.14 ± 0.33	3.68 ± 3.04
HIGH	0.11 ± 0.44	3.69 ± 6.26

Tabella 5.5: Medie e σ dei confronti tra le grandezze fittate e quelle originali delle tracce simulate per il tracking neurale.

File	$\bar{p_t}$	σ_{p_t}
LOW	-0.64 ± 0.1	1.80 ± 0.12
MED	-0.21 ± 0.08	1.91 ± 0.08
HIGH	-0.48 ± 0.09	2.00 ± 0.14
File	λ	σ_λ
LOw	0.09 ± 0.41	7.57 ± 0.44
MED	-0.08 ± 0.95	7.63 ± 0.44
HIGH	-0.15 ± 0.28	8.07 ± 0.33
File	ϕ	σ_ϕ
LOW	0.35 ± 0.69	7.87 ± 0.56
MED	0.36 ± 0.39	8.17 ± 0.38
HIGH	0.18 ± 0.26	8.34 ± 0.27

Tabella 5.6: Medie e σ dei confronti tra le grandezze fittate e quelle originali delle tracce simulate per il Kalman filter.

File	$\bar{p_t}$	σ_{p_t}
LOW	-1.12 ± 0.20	2.12 ± 0.22
MED	-0.90 ± 0.09	1.97 ± 0.07
HIGH	-1.16 ± 0.10	2.20 ± 0.09
File	λ	σ_λ
LOW	-0.46 ± 1.10	10.15 ± 1.24
MED	0.12 ± 0.66	9.87 ± 0.63
HIGH	-1.22 ± 0.55	10.81 ± 0.63
File	ϕ	σ_ϕ
LOW	-0.27 ± 0.42	5.92 ± 0.40
MED	0.06 ± 0.33	6.77 ± 0.34
HIGH	0.44 ± 0.30	7.56 ± 0.49

Tabella 5.7: Medie e σ dei confronti tra le grandezze fittate e quelle originali delle tracce simulate per l'FTF (caso 100×100).

Si potrà evincere dalle figure, come i risultati dell'algoritmo neurale siano molto incoraggianti di per sé, data la novità assoluta del metodo in questo campo specifico di applicazione, anche se il confronto quantitativo con il Kalman filter e con l'FTF vede prevalere questi ultimi in un rapporto che è circa 1.5:1, specie alle alte molteplicità di tracce. Ciò è prevedibile, dato lo stadio sperimentale a cui è il metodo proposto in questo lavoro di tesi. Il prossimo capitolo cercherà di illustrare quanto è possibile argomentare per giustificare quanto si osserva nei risultati, ed inoltre concluderà il presente studio con le riflessioni che questi risultati innescano allo scopo di proporre eventuali migliorie al sistema, in modo da valutarne le reali possibilità.

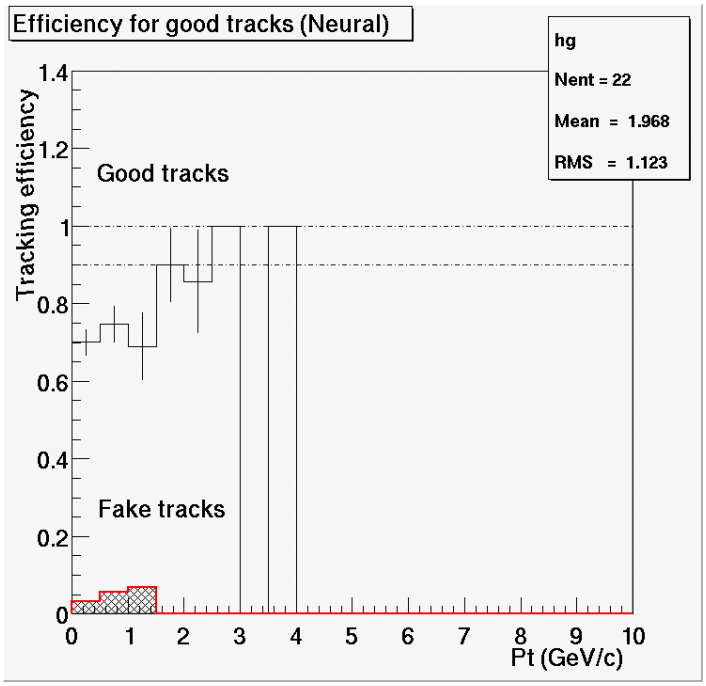

Figura 5.14: Efficienza per unità di impulso trasverso nel file a bassa molteplicità per il tracking neurale.

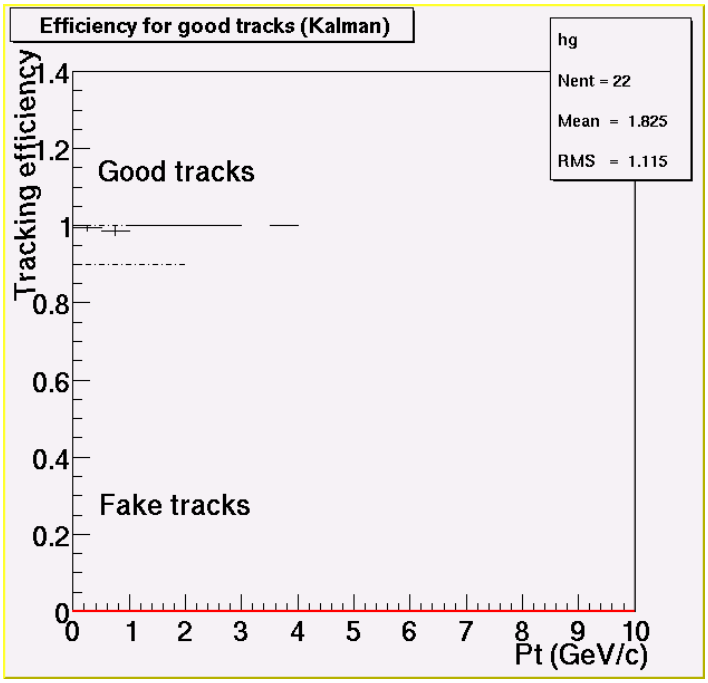

Figura 5.15: Efficienza per unità di impulso trasverso nel file a bassa molteplicità per il Kalman filter.

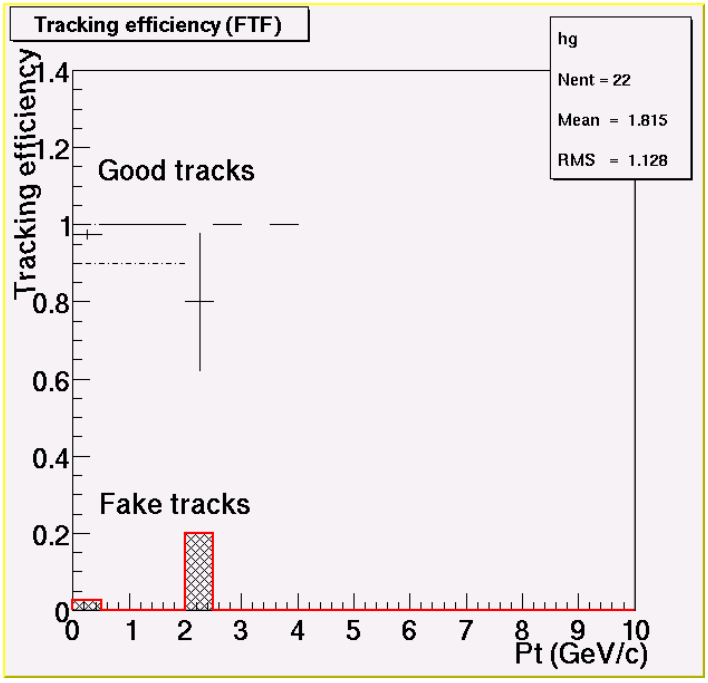

Figura 5.16: Efficienza per unità di impulso trasverso nel file a bassa molteplicità per l'FTF.

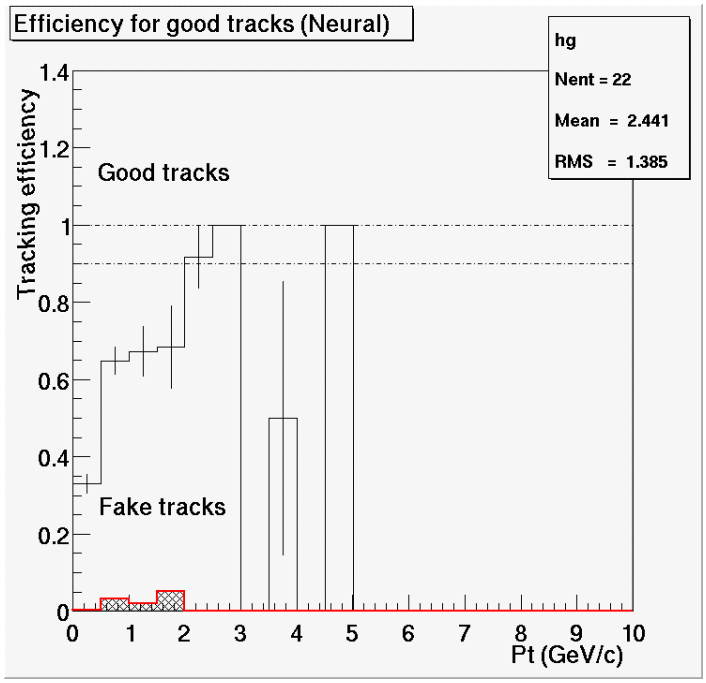

Figura 5.17: Efficienza per unità di impulso trasverso nel file a media molteplicità per il tracking neurale.

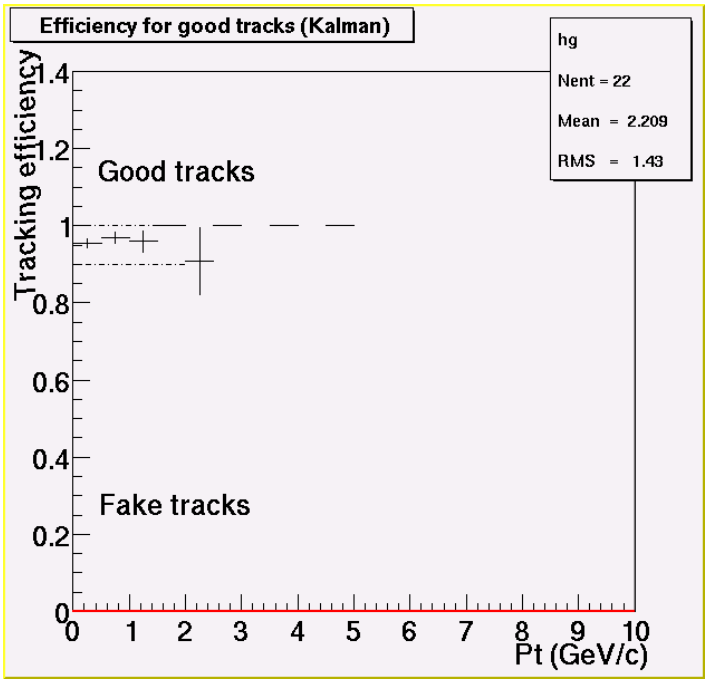

Figura 5.18: Efficienza per unità di impulso trasverso nel file a media molteplicità per il Kalman filter.

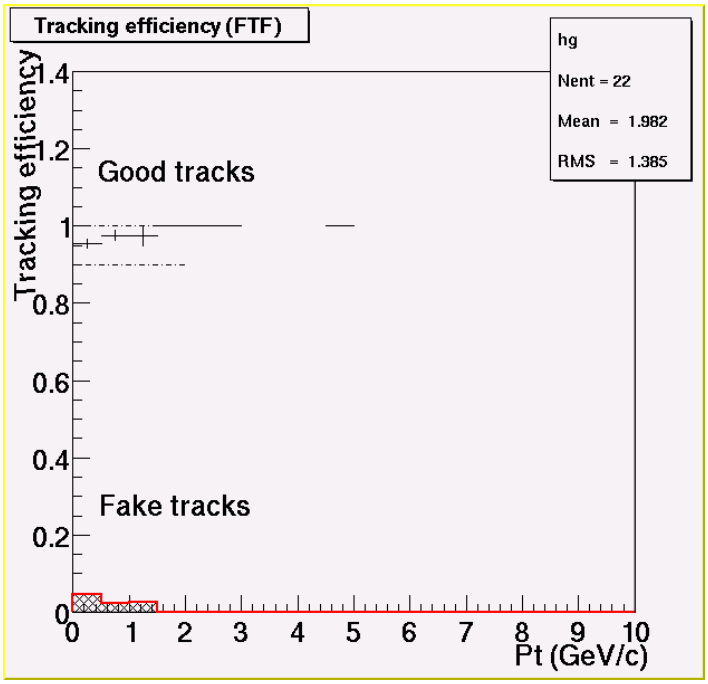

Figura 5.19: Efficienza per unità di impulso trasverso nel file a media molteplicità per l'FTF.

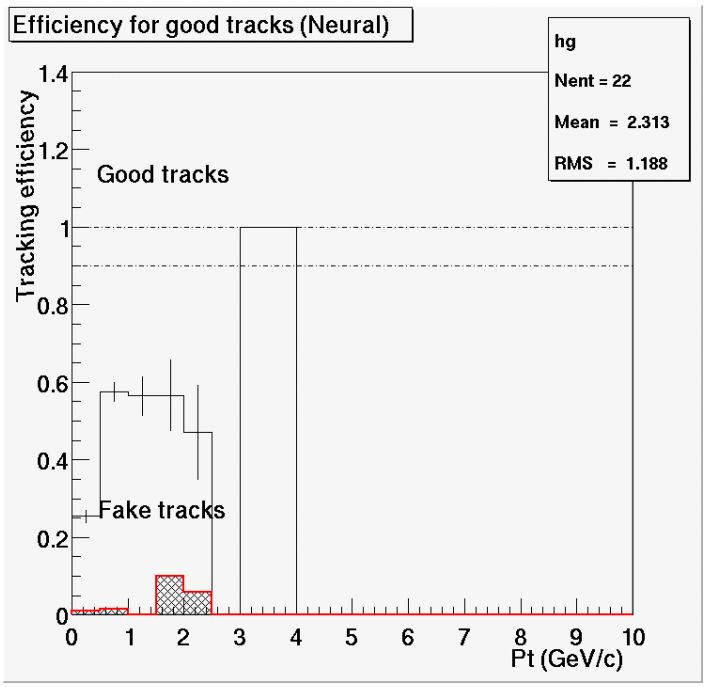

Figura 5.20: Efficienza per unità di impulso trasverso nel file ad alta molteplicità per il tracking neurale.

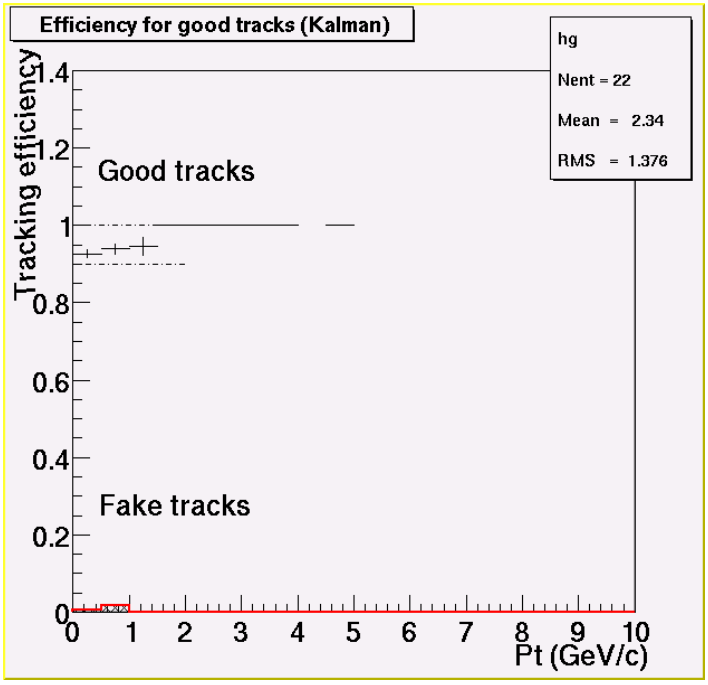

Figura 5.21: Efficienza per unità di impulso trasverso nel file ad alta molteplicità per il Kalman filter.

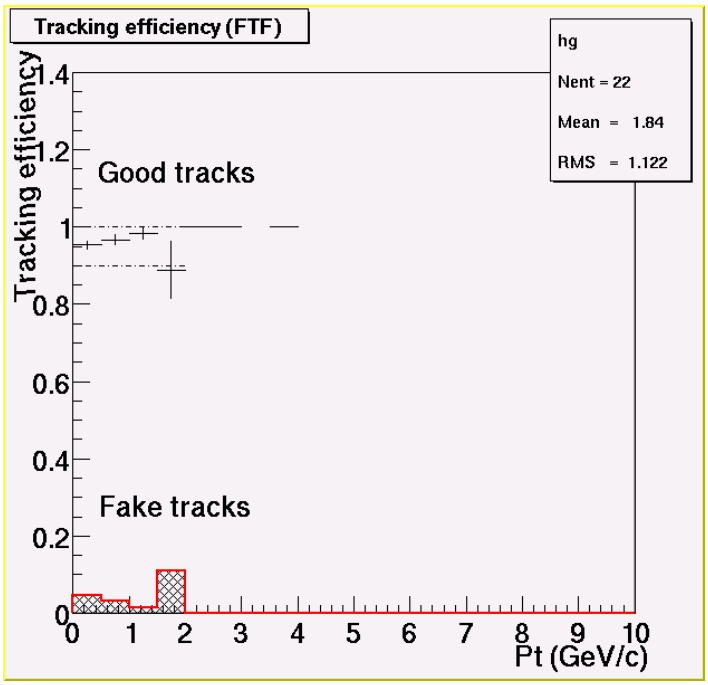

Figura 5.22: Efficienza per unità di impulso trasverso nel file ad alta molteplicità per l'FTF.

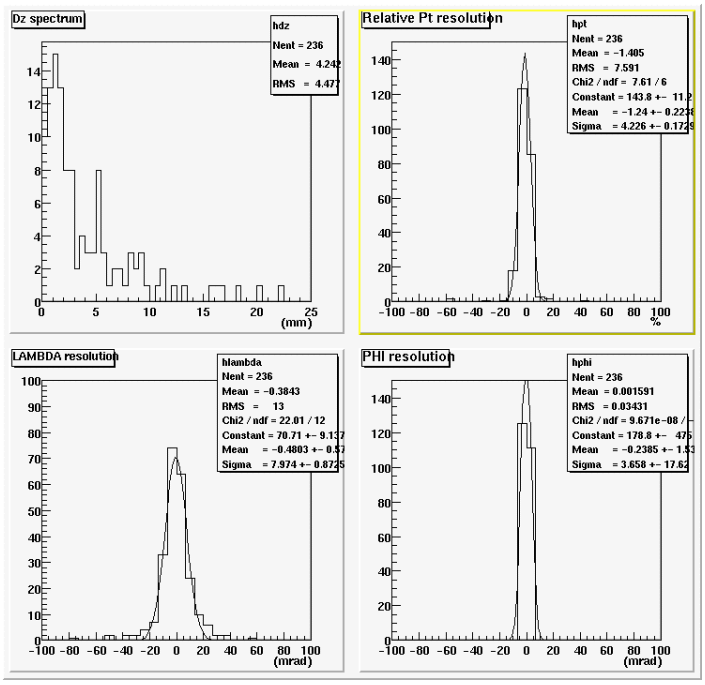

Figura 5.23: Fit dei parametri fisici dalle tracce buone trovate nel file a bassa molteplicità per il tracking neurale.

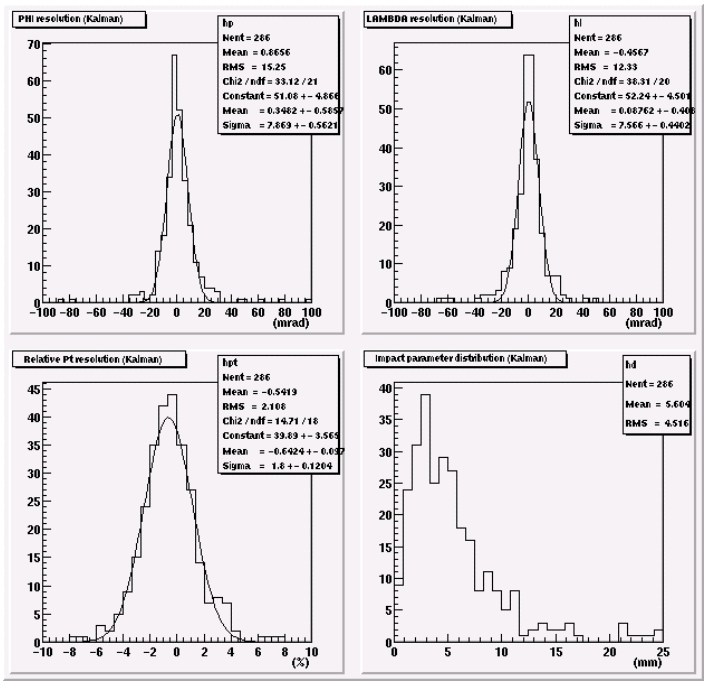

Figura 5.24: Fit dei parametri fisici dalle tracce buone trovate nel file a bassa molteplicità per il Kalman filter.

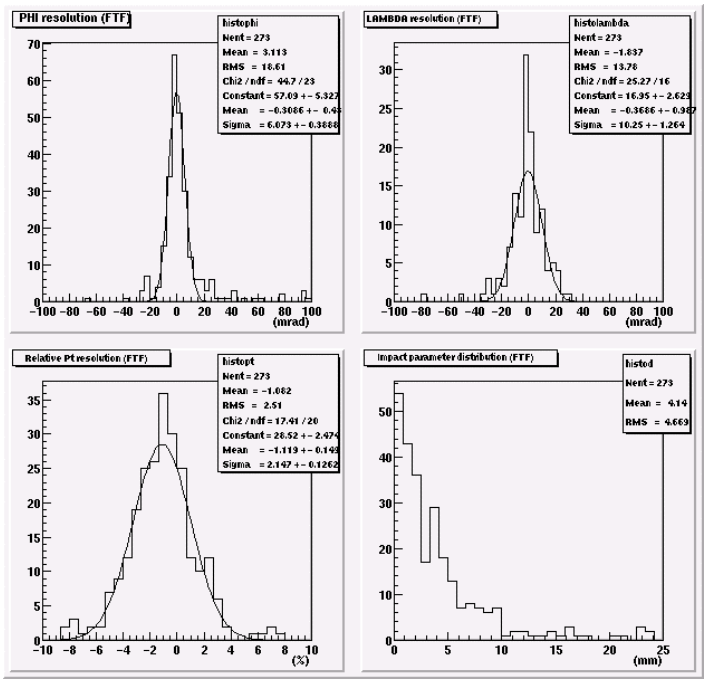

Figura 5.25: Fit dei parametri fisici dalle tracce buone trovate nel file a bassa molteplicità per l'FTF.

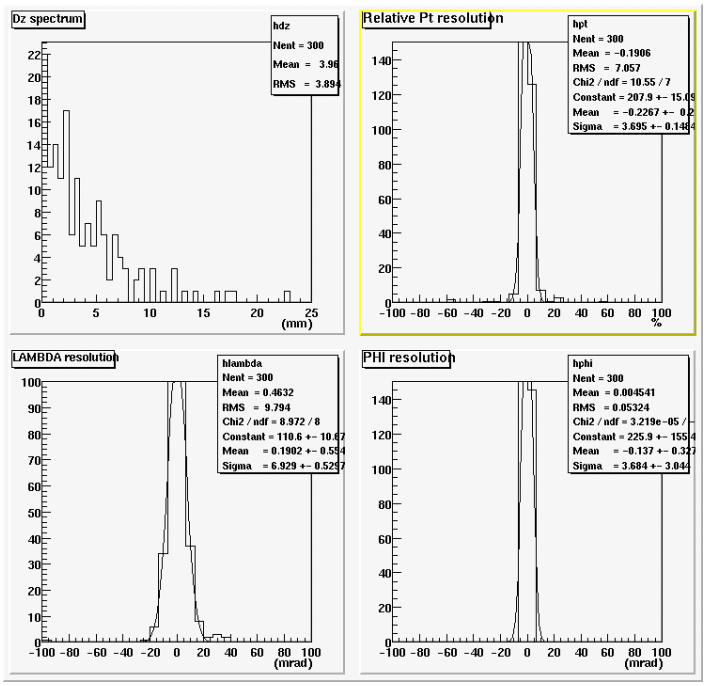

Figura 5.26: Fit dei parametri fisici dalle tracce buone trovate nel file a media molteplicità per il tracking neurale.

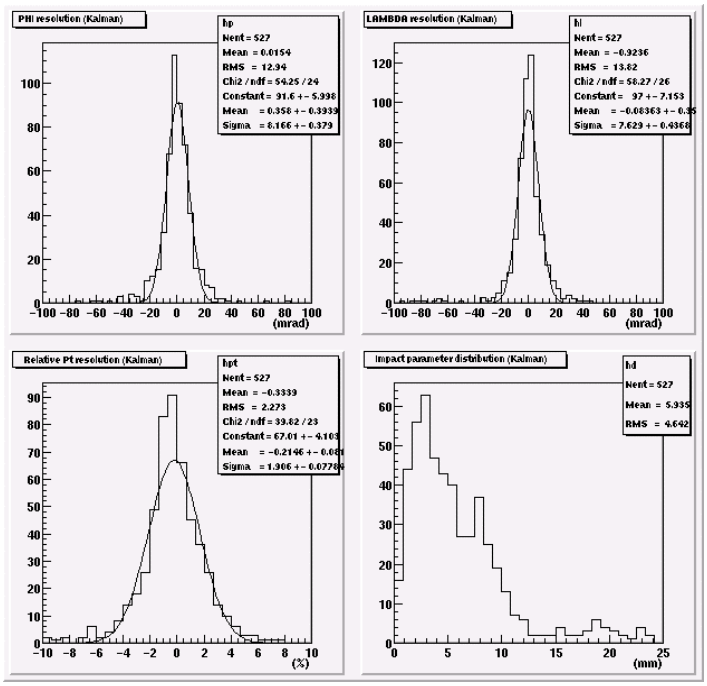

Figura 5.27: Fit dei parametri fisici dalle tracce buone trovate nel file a media molteplicità per il Kalman filter.

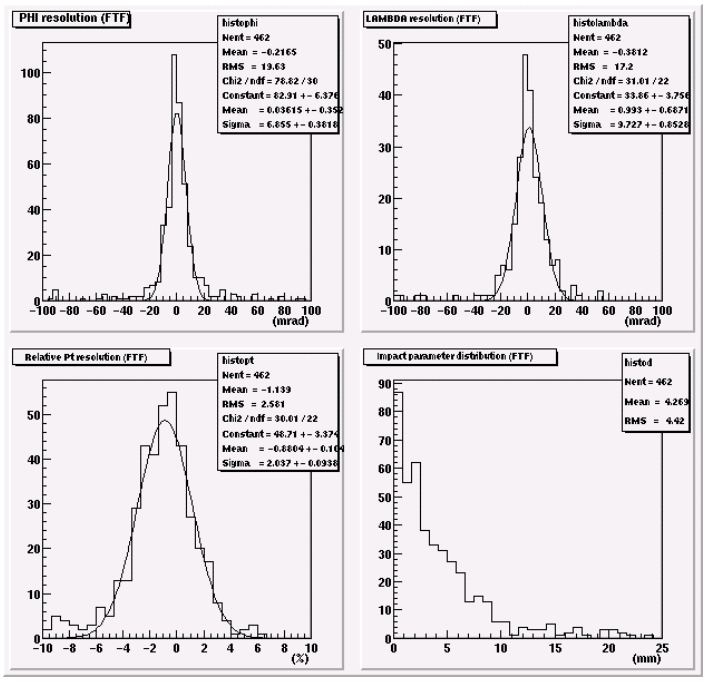

Figura 5.28: Fit dei parametri fisici dalle tracce buone trovate nel file a media molteplicità per l'FTF.

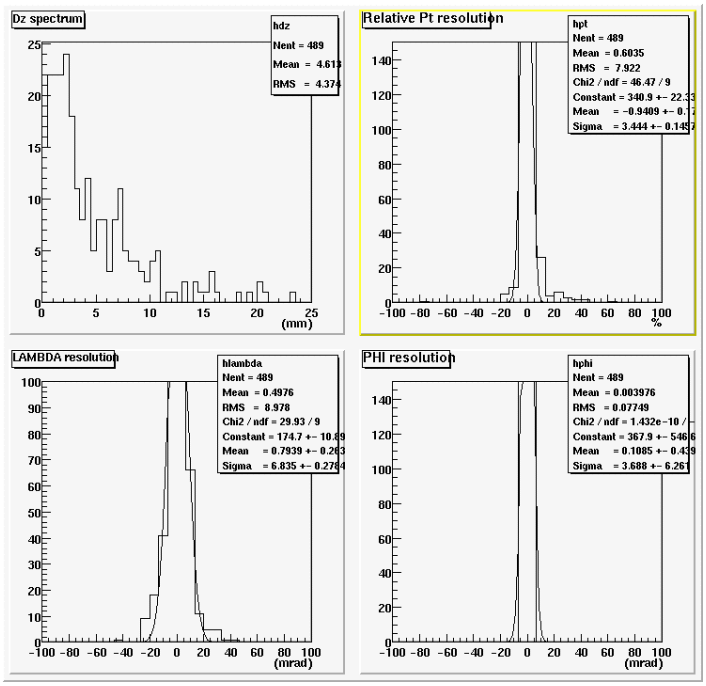

Figura 5.29: Fit dei parametri fisici dalle tracce buone trovate nel file ad alta molteplicità per il tracking neurale.

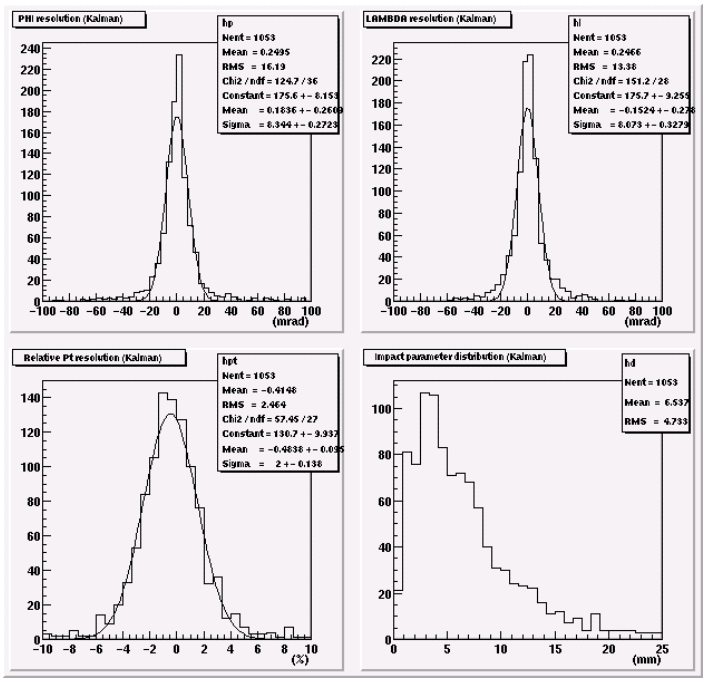

Figura 5.30: Fit dei parametri fisici dalle tracce buone trovate nel file ad alta molteplicità per il Kalman filter.

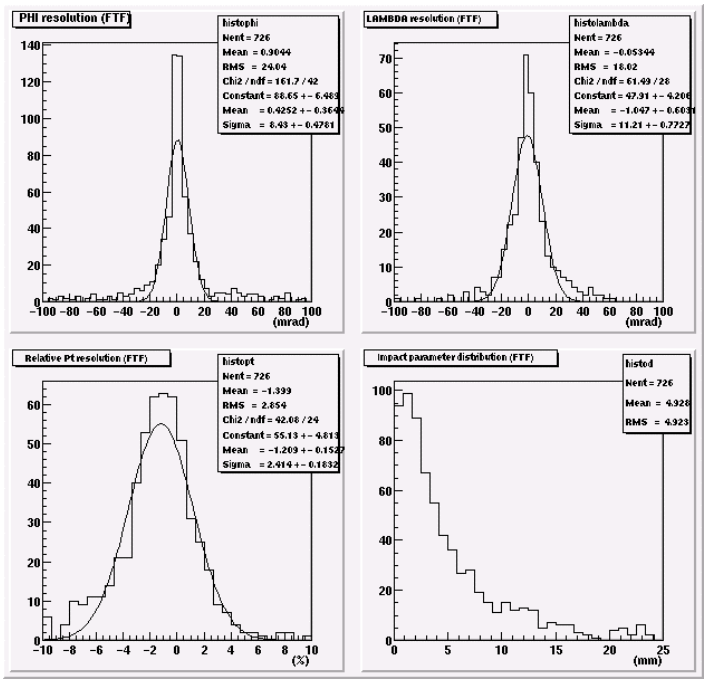

Figura 5.31: Fit dei parametri fisici dalle tracce buone trovate nel file ad alta molteplicità per l'FTF.

Capitolo 6

Conclusioni

Come è possibile notare, specialmente alle molteplicità media ed alta, l'efficienza dei risultati della rete neurale, così come si è riuscita ad implementare finora, non è assolutamente paragonabile a quanto prodotto dai metodi alternativi quali il Kalman filter e l'FTF. Le spiegazioni che è possibile trovare a questo fatto servono fondamentalmente a comprendere eventuali vie di sviluppo futuro che è possibile cercare di percorrere per migliorare le prestazioni della rete.

Anzitutto, è il caso di ribadire come la rete neurale ricostruisca la traccia in un modo "diretto", mentre, in genere, gli algoritmi consueti seguono un procedimento "graduale", che aggiunge un punto alla volta, scegliendolo in base all'accordo con quanto già ricostruito. Naturalmente, tale procedimento è in grado di controllare meglio situazioni che invece la rete deve prevedere. Per esempio, quando una traccia, come alcune che sono state prodotte nei file analizzati, non colpisce alcuni pad-row, si innescano una serie di situazioni per cui diventa difficile che la rete neurale, almeno con i pesi definiti dalla regola DPM, riesca a ricostruirla correttamente.

Intanto, può essere che il massimo consentito per la differenza di pad-row tra i due estremi di un neurone "costruibile" sia troppo basso perché vengano congiunti due punti consecutivi di una traccia che si comporti come sopra. Inoltre, specie nel caso dei piccoli impulsi trasversi, quando bisogna congiungere due punti che differiscono di diversi pad-row, l'orientamento del segmento è tanto più discosto dalla direzione radiale quanto più la differenza in pad-row è grande. Ciò, naturalmente, comporta che spesso due punti in queste condizioni non riescano a soddisfare i criteri di "cut" imposti sull'angolo azimutale. Queste sono situazioni in cui non si riesce a creare uno dei segmenti che sarebbero necessari affinché la traccia venga trovata e ciò pone un importante taglio all'efficienza. In questo caso, potrebbe essere una soluzione una sorta di cut dinamico, in cui la differenza di angolo azimutale tra gli estremi possa essere più o meno grande, secondo la differenza in pad.row tra i punti. Infatti, aumentare semplicemente l'accettanza angolare, nelle prove fatte, è servita soltanto ad aumentare considerevolmente il tempo di calcolo, senza condurre a vantaggi quantitativi apprezzabili. Al contrario, talvolta, la presenza, a causa di tale allargamento, di moltissime competizioni tra neuroni entranti nello stesso punto ed uscenti da esso, porta facilmente ad uno spegnimento dell'intera rete neurale, conseguenza ovvia della presenza, per ogni unità, di molte altre che contribuiscono ad un legame inibitorio. Va puntualizzato, in questo discorso, che si è provato a realizzare una procedura di aggiornamento "del vecchio tipo" ossia senza scegliere miglior precedente o miglior seguente, nella speranza che ciò incrementasse in modo sostanziale il contributo eccitativo, ma anche così in una situazione di eccessiva accettanza in ϕ, spesso la rete si è spenta tutta senza

restituire alcuna traccia. In ogni caso, allargare troppo il cut in ϕ ha ridotto il numero di tracce trovate. Naturalmente, tale discorso è meno importante per l'angolo polare θ, in quanto esso è meno influenzato dalla curvatura delle tracce, anche perché, nella situazione sperimentale di ALICE, il fatto che, preferenzialmente, le traiettorie delle particelle siano dirette in senso trasversale all'asse di collisione, che è poi anche l'asse del campo magnetico, fa sì che le eliche risultino estremamente schiacciate lungo l'asse di avvitamento. Allora, la variazione in angolo θ in una traccia è molto piccola anche tra punti di pad-row distanti (chiaramente entro differenze ragionevoli). L'unico caso che può comportare brusche variazioni in θ è un decadimento o uno scattering multiplo e qualche effetto del genere è stato in realtà riscontrato. Naturalmente, quando avvengono decadimenti o scattering multipli, si può ribadire quanto detto in precedenza, ossia che le tracce vengono divise in due, ma se il decadimento avviene nella zona centrale della TPC, c'è una buona probabilità che i due tronconi in cui la traccia si divide non siano ricostruibili perché contengono meno dei 32 punti previsti per il riconoscimento. A questo punto, chiaramente, si comprende come questi effetti fisici possano giocare un ruolo importante nell'efficienza della rete neurale.

Altro problema è insito nella definizione del peso sinaptico. Esso, come si è visto nel capitolo precedente, è correlato con la lunghezza del segmento e con l'allineamento di una coppia di segmenti. Si è detto, tra l'altro, che questi concetti fisici hanno senso solo quando i segmenti stessi sono consecutivi e brevi. D'altro canto, si potrebbe, nel caso in cui fosse necessaria una maggior precisione, sfruttare una definizione diversa di peso sinaptico. Esaminando per esempio l'algoritmo FTF [ALICE-Trk], si vede che esso si basa su un modello geometrico in base al quale ricostruire una traccia operando un best-fit delle grandezze fisiche che ne governano la forma. In quel caso si opera una trasformazione conforme della circonferenza, in modo da valutare quanto bene il sistema di punti trovati si adatta a quella forma geometrica. Anche nel caso del tracking neurale si può tentare una strada del genere, in cui si definisca come peso eccitativo, ad esempio, un valore correlato al risultato di un fit degli estremi dei neuroni con un cerchio, magari anche passante per l'origine, in modo da stabilire l'accordo tra elementi non tanto in base all'allineamento, quanto piuttosto in base ad una corrispondenza nell'ottica di vederli come archi di cerchio passanti per l'origine. Ancor meglio potrebbe essere fatto se si tenesse conto della perdita di energia, ma in effetti, salvo in casi in cui le particelle dovessero fermarsi nel mezzo, il che non succede, la perdita di energia in un tratto breve quanto può essere un neurone, non sembra sia passibile di implicazioni talmente influenti da condurre al rischio di errori gravi se la si trascura.

Naturalmente, in questo tipo di progetto, andrebbe ridefinito anche l'insieme dei cut, in quanto, a questo punto, bisognerebbe mettere tutte le caratteristiche di un neurone in relazione al cerchio che lo attraversa. Ad esempio, non avrebbe più tanto senso come segmento, bensì come arco di cerchio, e quindi non conterebbero più tanto le coppie di neuroni, bensì le terne, dato che tre sono i punti che individuano in modo univoco un arco di cerchio. Oppure, sempre nell'ambito del vertex constraint, si potrebbero ancora considerare le coppie ma includendo nell'insieme l'origine, il che comporterebbe la possibilità di definire un centro ed un raggio di curvatura assoluto per ogni coppia di punti (calcolabile geometricamente), e impostare i confronti tra unità che conducono alla definizione dei pesi sinaptici al raffronto di questi elementi geometrici. Tutti questi possibili spunti di elaborazione successiva non sono stati sviluppati nella tesi in quanto necessitano di un'accurata analisi qualitativa, per valutare il modo di implementarne i concetti in uno schema di RNA e poi quello di comporli in un programma. In realtà questo lavoro è voluto essere il test di un metodo proposto, in un recente passato, per esperimenti di ricostruzione di tracce, ma

mai applicato a situazioni così "difficili" come quella di ALICE. Infatti, facendo delle prove con file di dati contenenti meno tracce e con molteplicità molto minore, in genere l'efficienza del sistema è risultata ben migliore (un file di prova con solo 30 tracce in tutto lo spazio è stato perfettamente ricostruito !). In base a quanto osservato, gli sviluppi successivi a cui questo lavoro di tesi apre la strada vanno essenzialmente in quattro direzioni:

- anzitutto, la ricerca di configurazioni migliori per ottimizzare i criteri di elaborazione e, se possibile, superare i criteri dell'array per giungere alla gestione diretta degli indirizzi di memoria (usando le liste concatenate citate nel capitolo 5);

- migliorare i criteri di scelta dei neuroni; osservando ancora una volta il modo di operare dell'FTF, si può notare come esso divida lo spazio in sub-volumi in modo da dare un criterio alle regioni in cui scegliere il punto successivo, volta per volta, della traccia ricostruita; se si riescono a trovare criteri di corrispondenza alla forma prevista per una traccia più restrittivi di un cut sugli angoli e sulla differenza di pad, si potrebbe rendere la routine di creazione dei neuroni più efficiente, in quanto creerebbe con molta probabilità tutti i neuroni "giusti" e ne creerebbe di "sbagliati" in numero abbastanza piccolo da non causare eccessiva confusione, cosa che in effetti è il motivo di performance scadenti quando si allargano le accettanze, come si è detto sopra;

- creare relazioni concettuali migliori e definire pesi sinaptici più specifici per una traccia elicoidale, che permettano di selezionare con maggior efficienza i punti correttamente associati e, contemporaneamente, di gestire meglio eventuali incroci e biforcazioni;

- estendere l'analisi neurale agli altri elementi dell'intero sistema ALICE, in particolare l'ITS che, similmente alla TPC, restituisce diversi punti su diversi strati sensibili nella regione più interna.

Naturalmente questo implica una serie di accorgimenti per garantire il "matching" tra rivelatori e mantenere i miglioramenti alle risoluzioni sperimentali conseguite da questi dispositivi, il che si può garantire con una notevole efficienza non solo dal punto di vista del numero di tracce buone trovate, ma anche e soprattutto dal punto di vista dei punti in ogni traccia; si ricordi che una traccia può essere "buona" anche se contiene meno punti di quanti dovrebbe e con alcuni errori, ma queste situazioni comporterebbero un peggioramento della qualità dei best-fit e quindi errori sui parametri fisici notevoli. Questi sono quindi i risultati e le possibili proposte di sviluppo futuro della rete neurale per il tracciamento in ALICE. Il tempo a disposizione potrebbe condurre ad un'efficienza maggiore, ma in ogni caso, con questo lavoro, si è conseguito lo scopo di applicare un metodo innovativo ad un problema di Fisica molto arduo, il che è senza dubbio uno sviluppo importante della tecnologia delle reti neurali e mostra le possibilità che tali sistemi aprono nell'analisi dei dati in Fisica Nucleare e Subnucleare, la quale, come molti esperimenti hanno dimostrato, non può che trarre giovamento da quanto di più efficace l'informatica e l'automazione del calcolo possono introdurre.

Appendice A

Convenzioni di metalinguaggio

A.1 Convenzioni tipografiche

Anzitutto, è necessario esprimere alcune convenzioni tipografiche del metalinguaggio. A questo scopo è preposto il seguente schema:

Scrittura	Significato	Esempio
`MAIUSCOLO`	parola chiave (invariabile) di metalinguaggio	`INT`
`normale`	nome variabile	`INT i`
corsivo	segnaposto	`INT` *var_intera*
`[espressione]`	specifica facoltativa	`INT i[=5]`
`{opz1|opz2}`	scelta fra opzioni	`DO {WHILE|UNTIL}`
`underscore ('_')`	interruzione e prosecuzione della riga di comando a capo	

A.2 Tipi di variabile

La sintassi di una dichiarazione di variabile è sempre della forma seguente:

tipo_variabile nome_variabile `[=` *valore*`];`
tipo_variabile`[]` *nome_variabile*;

in cui *tipo_variabile* è il tipo dei dati previsto e *nome_variabile* è il nome della variabile definito dall'utente. Se si pospongono delle parentesi quadre `[]` ad un nome di variabile o al suo tipo nella dichiarazione, si vuole indicare non una struttura facoltativa, ma un array di valori del tipo definito. La parte facoltativa presente nell'espressione, viene usata quando si vuole fissare immediatamente il valore della variabile, nel caso di una dichiarazione singola. Nonostante gli standard di molti linguaggi lo prevedano, in nessuna parte del testo si è specificata la dimensione di un array, perché questa dipende dall'applicazione specifica. Piuttosto, si suppone che sia presente una variabile intera contenente tale dimensione in un valore opportuno, che non si presuppone ottenibile direttamente dall'array stesso. I tipi di variabile possono essere quelli standard, presenti nei linguaggi comuni, oppure strutture di dati più complicate, definite dall'utente come combinazioni degli stessi. I tipi standard sono i seguenti:

`INT`	*variabile intera*
`REAL`	*variabile reale*
`BOOL`	*variabile booleana (vero/falso)*
`CHAR`	*carattere*
`STR`	*stringa di caratteri*

È il caso di sottolineare come molti linguaggi di programmazione possiedano più tipi interi e più decimali, distinti dal numero di byte disponibili per memorizzare il valore. Tuttavia, non è opportuno, in una descrizione non specifica quale questa vuole essere, distinguere gli interi in "lunghi" e "corti" o i reali in "a precisione singola" ed "a precisione doppia", come si usa in informatica. Sarà il programmatore a stabilire quale dei due tipi è più opportuno all'applicazione realizzata. Per indicare la definizione di una struttura di dati, si usa la seguente forma:

```
STRUCTURE nome {
   tipo1 nome1;
   tipo2 nome2;
   ...
   tipoN nomeN;
};
```

in cui esistono diversi campi, che racchiudono valori associati allo stesso elemento. Per richiamare poi il valore *nomeN* associato ad una variabile dichiarata del tipo *nome*, con il nome, ad esempio, `V1`, si userà la scrittura `V1.`*nomeN*, in cui il nome a sinistra del punto è proprio quello della variabile e il termine a destra è il nome del campo, così come compare nell'istruzione `STRUCTURE`.

A.3 Strutture modulari

Volendo scrivere un metalinguaggio comprensibile ma non troppo specifico, si è pensato di esprimere qualunque struttura modulare con lo stesso schema, del tipo:

```
tipo_modulo (argomenti)
   ⋮
   istruzioni
   ⋮
END tipo_modulo
```

in cui, mediante l'uso dell'indentazione, si cerca di rendere chiari i limiti di ciascun modulo. Gli elementi essenziali, quindi, sono due: il tipo di modulo e gli argomenti. Di conseguenza, in seguito, si specificheranno solo le sintassi della riga del titolo, con il tipo di modulo e la forma degli argomenti, dando per scontato che segua una sequenza di istruzioni all'interno e sia specificato, ove serve, un "nome proprio". È previsto l'uso di due livelli di strutture modulari: il più alto è quello della *subroutine* e della *funzione*, che definiscono sequenze strutturate di operazioni, definite in un punto del programma, e poi eseguite ogni qualvolta le si richiama, menzionandone il nome ed eventuali argomenti. La differenza è che la subroutine non restituisce un valore, mentre la funzione lo restituisce. In questo caso, gli argomenti sono normali dichiarazioni di tipo, che fungono da segnaposto per i valori

che verranno immessi da programma, e che determineranno le modalità delle procedure indicate nel modulo. La parola-chiave usata nella definizione delle subroutine è SUB e quella adoperata per le funzioni è FUNC: quest'ultima deve essere seguita da un tipo dati, che è quello del valore restituito. Ecco la forma delle righe del titolo per la definizione di queste strutture modulari:

```
FUNC {INT|REAL|CHAR|STR|struttura} nome (argomenti)
   ⋮
   istruzioni
   ⋮
   RETURN valore;
   ⋮
END FUNCTION

SUB nome (argomenti) ...END SUB
```

Si è esplicitata la struttura modulare di FUNC perché vi deve comparire necessariamente l'istruzione RETURN, la quale indica il punto in cui viene definito il valore da restituire. Si presuppone che quando l'esecuzione giunge ad una istruzione RETURN, il programma esce dalla funzione e restituisce il valore indicato.

Ad un livello più basso, si sono previsti tre tipi di modulo: l'iterazione, la decisione e la ripetizione condizionata. Nell'ordine, questi tre tipi di struttura sono espressi dalle parole chiave FOR, IF...ELSE...END IF e DO. Il ciclo FOR prevede l'esecuzione ripetuta del modulo in base al valore di una variabile, sotto indicata con *var_incr*, che viene inizializzata la prima volta al valore *val_iniz*, e incrementata ogni volta di 1, se il quarto argomento manca, oppure di una quantità uguale ad esso, se è presente. Quando la variabile *var_incr* diventa più grande di *val_fin*, le ripetizioni terminano. Se si specifica *val_fin* minore di *val_iniz*, definendo un incremento negativo, il processo funziona all'inverso. La sintassi è la seguente:

```
FOR (var_incr, val_iniz, val_fin, [incr])
```

Quanto al ciclo IF, esso racchiude un modulo che viene eseguito una volta sola, ma unicamente nel caso in cui una condizione, espressa da una scrittura che possa essere "vera" o "falsa" (esempio, una diseguaglianza come 'i < 0'), risulta "vera". È possibile specificare un modulo alternativo, che indichi al programma cosa fare in entrambi i casi, mediante la parola-chiave ELSE:

```
IF (condizione) ... [ELSE ...]  END IF
```

Infine, il ciclo DO impone di ripetere continuamente una serie di istruzioni, senza alcun controllo. Si può comunque porre un'istruzione WHILE o UNTIL, seguite da una condizione analoga a quella che compare in IF, la quale viene controllata alla fine di ogni ciclo, di modo che, se è vera, WHILE impone di ripetere il ciclo, mentre UNTIL impone di terminarlo, mentre se è falsa, le istruzioni si comportano esattamente al contrario. La sintassi usata, quindi, è la seguente:

```
DO [{UNTIL|WHILE (condizione)}] ... END DO
```

Si può applicare alle condizioni la logica booleana, utilizzando, per combinarle, gli operatori binari `AND`, `OR` e `NOT`. Inoltre, nell'eventualità di dover uscire da un modulo "anzitempo", in seguito al verificarsi di situazioni particolari, è previsto l'uso del comando `BREAK`. Per indicare, invece, che si intende saltare solo l'esecuzione del ciclo in fase di elaborazione, ma se ne vuole far partire un altro dopo aver aggiornato le variabili interessate, si userà il comando `CONTINUE`.

A.4 Strutture dati di comune utilità e funzioni predefinite

Questa tesi non vuole essere un trattato di informatica né presentare in estremo dettaglio i criteri di strutturazione di uno qualunque dei file discussivi. Non è suo scopo esplorare troppo specificamente la strutturazione di ogni particolare della programmazione, ma solo presentare un modello algoritmico per la realizzazione di simulazioni al calcolatore di elaborazioni neurali. In quest'ottica, non si è rivelato utile, né vantaggioso addentrarsi nei dettagli della definizione di diverse strutture di dati che possono servire, come anche di molte funzioni. Esse, quando si rivelassero necessarie, saranno presentate lungo il testo, e ivi definite. Esempio di quanto detto sono alcune funzioni che operano semplici calcoli matematici, come la funzione `ABS`, che restituisce il valore assoluto del suo argomento ed `EXP`, che calcola l'esponenziale dell'argomento, e alcune *shortcut* per alcune operazioni ripetute (per esempio la scrittura `X += value` al posto di `X = X + value`). Esse vengono specificate nel testo, nei punti in cui si presentano, allo scopo di conseguire una maggiore immediatezza, evitando la necessità di consultare l'appendice ad ogni riga, il che renderebbe dispersiva e frammentaria la lettura di qualunque esempio implementativo presente nella tesi.

Appendice B

Calcoli per il best-fit elicoidale

Quest'appendice ha due scopi:

1. illustrare le grandezze fisiche che vengono misurate in ogni traccia trovata dalla rete neurale;
2. proporre i calcoli di best-fit eseguiti per determinare tali grandezze in base alle terne di punti ottenuti dall'analisi neurale.

Per questo, si è scelto di suddividerla in due paragrafi, ciascuno dei quali esaurisce uno di questi punti.

B.1 Grandezze fisiche da misurare

Il punto di partenza è la formula della forza di Lorentz, che è il caso di esprimere in forma relativistica, date le condizioni sperimentali in questione:

$$\mathbf{F} = \frac{e}{\gamma m}\mathbf{p} \times \mathbf{B} = \frac{eB}{\gamma m}(p_y\mathbf{i} - p_x\mathbf{j}) \tag{B.1}$$

in cui, come nella notazione consueta, $\gamma = \sqrt{1 - \frac{v^2}{c^2}}$ e m è la massa a riposo della particella, la quale si assume abbia carica e. L'ultimo membro è legittimato dalla supposizione, fatta nel resto della tesi, che il campo magnetico sia parallelo all'asse z. Le equazioni del moto per la particella in questione, dalle tre componenti della (B.1) sono le seguenti:

$$\omega(p_y\mathbf{i} - p_x\mathbf{j} + 0\mathbf{k}) = \dot{p}_x\mathbf{i} + \dot{p}_y\mathbf{j} + \dot{p}_z\mathbf{k} \tag{B.2}$$

in cui $\omega = eB/\gamma m$. Le soluzioni di questa terna di equazioni si possono esprimere con la notazione seguente, in cui vengono incluse le opportune costanti arbitrarie:

$$\begin{cases} p_x(t) = p_t \cos(\omega t + \alpha) \\ p_y(t) = -p_t \sin(\omega t + \alpha) \\ p_z = \text{cost.} \end{cases} \tag{B.3}$$

Delle due costanti arbitrarie che compaiono nella (B.3), p_t rappresenta il modulo della proiezione dell'impulso totale $\mathbf{p}$ sul piano xy, che quindi è costante ed è noto come *impulso trasverso*. Il fattore di fase, invece, può essere dedotto dal valore del vettore

impulso totale nel punto di partenza della traiettoria, ossia nel *vertice* dell'evento. Ricordando, poi, che l'impulso vettoriale relativistico è γmv, è possibile dedurre dalle (B.3) le equazioni differenziali dalle quali si ricavano quelle parametriche della curva descritta dal moto della particella. Queste ultime risultano essere le seguenti, ove si è posto $b = p_z/\gamma m$ e $R = p_t/\gamma m\omega$:

$$\begin{cases} x(t) = x_0 + R\sin(\omega t + \alpha) \\ y(t) = y_0 + R\cos(\omega t + \alpha) \\ z(t) = bt + z_{in} \end{cases} \tag{B.4}$$

I valori x_0, y_0 e z_{in} sono costanti arbitrarie necessarie come è noto dalla teoria delle equazioni differenziali di primo ordine. È evidente che questa curva è un'elica con l'asse di avvitamento parallelo a z, e che la sua proiezione sul piano xy è proprio una circonferenza di raggio R e centro (x_0, y_0), mentre z_{in} è la coordinata z del punto occupato nell'istante iniziale, secondo la scala di tempi utilizzata. Ciò giustifica la scelta di indicare con pedici diversi le costanti arbitrarie nella x e nella y rispetto alla z, visto il significato fisico differente delle tre. Dal momento che l'impulso trasverso della particella è una delle grandezze che occorre misurare, servirà stimare il raggio della traccia per ottenerlo, e ciò comporta la necessità di un best-fit della circonferenza ottenuta come proiezione dell'elica sul piano xy. È importante sapere anche com'è orientato l'impulso totale nel vertice della traccia, e quest'informazione viene dedotta da due angoli correlati con la sua direzione e verso in questo punto:

- l'angolo azimutale, indicato con ϕ ;
- il complementare dell'angolo polare, indicato con λ;

È immediato provare che $\tan\phi = p_y/p_x$ e che $\tan\lambda = p_z/p_t$.

Un'altra informazione che va calcolata a proposito del vertice è la sua distanza dall'origine. Infatti, non è necessariamente coincidente con essa, a causa delle indeterminazioni ed alla possibilità di aver incluso nella traccia ricostruita una seppur minima percentuale di punti sbagliati (si è detto, meno di 8). In realtà, il simulatore usato [GEANT3] è stato impostato in modo da produrre tracce che partissero esattamente dall'origine, per cui si è scelto, allo scopo di semplificare il fit, di utilizzare una condizione tipica di questo tipo di analisi: il "vertex constraint". Tale condizione impone di far passare la curva fittata per l'origine, il che comporta alcune restrizioni sui valori delle costanti che risultano dal fit. In particolare, il metodo che si è usato per ricostruire la proiezione circolare lungo il piano xy, necessita di un punto noto di tale circonferenza, e, con questa condizione, lo si pone uguale all'origine. Invece, la condizione non viene imposta nel fit spaziale conclusivo per la traccia, in modo che sia possibile avere una situazione in cui, quando la proiezione della traccia passa per l'origine in xy, non necessariamente la z si deve annullare. In questo modo, è importante conoscere la distanza tra questo punto e l'origine nello spazio, distanza che coincide proprio con la coordinata z del primo. Tale valore è indicato come *parametro d'impatto longitudinale* ed è l'ultimo valore fisico che si è avuta cura di stimare dalle tracce buone trovate. Naturalmente, comunque, è bene che tale valore sia piccolo (in genere il potere risolutivo della TPC è tale da condurre a valori contenuti entro l'ordine di grandezza del millimetro).

Riassumendo in un piccolo schema, ecco i valori da calcolare:

Parametro	Simbolo
impulso trasverso	p_t
parametro d'impatto longitudinale	D_z
angolo azimutale dell'impulso iniziale	ϕ
complementare dell'angolo polare iniziale	λ

B.2 Calcoli di best-fit

Noti che siano i parametri di interesse fisico da calcolare, si procede adesso ad illustrare i calcoli mediante i quali si sono ottenuti tali valori. Presupposto della discussione è che si parte da una serie di terne di coordinate, che sono quelle dei punti trovati in ogni traccia. Anzitutto, ricordando le definizioni date per ω nella (B.3) e per R e b nella (B.4), è possibile, mediante una serie di passaggi, riscrivere l'elica (B.4) nella forma seguente:

$$\begin{cases} x(t) = x_0 + R\sin(s+\alpha) \\ y(t) = y_0 + R\cos(s+\alpha) \\ z(t) = z_{in} + R\tan\lambda s \end{cases} \tag{B.5}$$

in cui $s = \omega t$.

Se inoltre si considera che l'istante iniziale corrisponde a $s = t = 0$, sostituendo tale valore nelle (B.3) si ottiene sia la fase α che l'angolo ϕ che è interesse presente ricavare. Sempre con lo stesso valore di s, sostituito stavolta nella terza equazione (B.5), si comprende che z_{in} è proprio il parametro d'impatto longitudinale di cui si è discusso. Allora, per maggior chiarezza, è possibile riscrivere definitivamente la (B.5) in funzione dei soli parametri di interesse:

$$\begin{cases} x(t) = x_0 + R\sin(s+\phi) \\ y(t) = y_0 + R\cos(s+\phi) \\ z(t) = D_z + Rs\tan\lambda \end{cases} \tag{B.6}$$

Il procedimento di best-fit opera in due fasi distinte: prima viene trovato, mediante un fit nel piano xy, il centro e il raggio della proiezione circolare dell'elica su di esso. Poi, a partire anche dai valori trovati nel passaggio precedente, viene fittata la terza equazione (B.6), dopo aver operato alcune sostituzioni grazie alle quali ricavare s dalle coordinate consenta di mantenerne la forma lineare.

B.2.1 Fit sul piano xy: trasformazione conforme

Usando la notazione proposta finora, imporre che la proiezione circolare della traccia passi per l'origine comporta un'importante semplificazione dell'equazione generica della circonferenza nel piano xy:

$$x(x - 2x_0) + y(y - 2y_0) = 0 \tag{B.7}$$

Il procedimento per fittare una circonferenza generica sul piano, è abbastanza complesso, non essendo risolubile analiticamente in modo diretto. Tra i diversi procedimenti possibili, allo scopo di fare un'analisi rapida con una precisione soddisfacente, si è scelto di sfruttare un criterio già proposto per un sistema di tracciamento veloce in ALICE [ALICE-FTF, FTF]. Questo metodo si basa sull'uso di una rappresentazione conforme, che consente di convertire, previo cambio di coordinate opportuno, l'equazione di una circonferenza in quella di una retta, per la quale il best-fit è una procedura standard, veloce

da eseguire. L'unica assunzione aggiuntiva di cui si necessita per procedere ai calcoli che questo metodo richiede, è quella di conoscere un punto appartenente alla traccia. La rappresentazione conforme, infatti, trasforma le coordinate cartesiane in coordinate prese in rapporto alla distanza di un punto generico della circonferenza da quello considerato noto. Imporre il vertex constraint, per quanto detto prima, significa fissare tale punto noto nell'origine. Se in generale, detto (x_t, y_t) il punto "noto" in uso, la trasformazione è la seguente:

$$\xi = \frac{x - x_t}{r^2} \qquad \text{ed} \qquad \eta = \frac{y_t - y}{r^2} \tag{B.8}$$

con $r^2 = (x - x_t)^2 + (y - y_t)^2$, allora, visto che in questo caso $x_t = y_t = 0$, sarà:

$$\xi = \frac{x}{r^2} \qquad \text{ed} \qquad \eta = -\frac{y}{r^2} \tag{B.9}$$

con $r^2 = x^2 + y^2$.

Operando la trasformazione delle coordinate nella (B.7), si ottiene:

$$\begin{gathered} r^2\xi(r^2\xi - 2x_0) - r^2\eta(-r^2\eta - 2y_0) = 0 \\ \Downarrow \\ r^4(\xi^2 + \eta^2) + 2r^2(y_0\eta - x_0\xi) = 0 \end{gathered} \tag{B.10}$$

Adesso, dalle sostituzioni (B.9), è evidente che $r^4(\xi^2 + \eta^2) = r^2$. Sostituendo questo risultato e semplificando il fattore comune r^2, si ottiene un'equazione lineare:

$$\eta = \frac{x_0}{y_0}\xi - \frac{1}{2y_0} \tag{B.11}$$

Se si pone, adesso, di ricavare le coordinate conformi dalle x e y dei punti trovati, e fittare un'equazione lineare del tipo:

$$\eta = A\xi + B \tag{B.12}$$

sarà, in base alla (B.11) e facendo le dovute corrispondenze con la (B.12):

$$\begin{cases} x_0 = -\frac{A}{2B} \\ y_0 = -\frac{1}{2B} \end{cases} \tag{B.13}$$

In conclusione, da questo primo passaggio si ottengono i valori di x_0, y_0 ed R, che si ottiene banalmente da $R = \sqrt{x_0^2 + y_0^2}$. Questi possono essere sfruttati, immediatamente per determinare l'impulso trasverso. Inoltre, nella posizione in cui al tempo iniziale la curva passa per il punto che si intende considerare il suo vertice, la fase ϕ è immediatamente determinabile da questi risultati. Infatti, calcolando il valore di x e della y, con le costanti ricavate ed al tempo $t = 0$, si ha:

$$\begin{cases} 0 = x_0 + R\sin\phi \\ 0 = y_0 + R\cos\phi \end{cases} \tag{B.14}$$

da cui discende immediatamente che $\tan\phi = \frac{x_0}{y_0}$.

Ultima cosa da notare è che i valori trovati servono anche a semplificare il best-fit successivo.

B.2.2 Fit nello spazio

Anziché sostituire il valore di s ottenibile risolvendo una delle prime due equazioni (B.6) soltanto, il che comporta l'impiego della funzioni arcoseno o arcocoseno, che non sono semplici da gestire e limitate in intervalli che potrebbero, in un'analisi al computer, comportare errori di calcolo o necessità di controlli sui dati in ingresso, si procede come segue:

$$\begin{cases} x - x_0 = R\sin(s+\phi) \\ y - y_0 = R\cos(s+\phi) \end{cases} \tag{B.15}$$

Il che consente di scrivere:

$$s = \arctan\frac{x-x_0}{y-y_0} - \phi \tag{B.16}$$

Allora, con questa sostituzione, la terza equazione della (B.6) diventa:

$$z = D_z + R\tan\lambda\left(\arctan\frac{x-x_0}{y-y_0} - \phi\right) \tag{B.17}$$

A questo punto si fa la seguente sostituzione:

$$\sigma = R\left(\arctan\frac{x-x_0}{y-y_0} - \phi\right) \tag{B.18}$$

in cui la variabile σ è determinabile, dato che le costanti in essa presenti sono state già ricavate. Allora l'equazione si riconduce ad una nuova retta da fittare:

$$z = D_z + C\sigma \tag{B.19}$$

da cui si ricava immediatamente D_z e, altrettanto rapidamente, si deduce λ, in quanto è evidente che, date queste impostazioni dell'equazione, è

$$C = \tan\lambda$$

In questo modo, tutte le grandezze di interesse fisico che si intendeva conoscere possono essere determinate.

Appendice C

Teorema di convergenza del percettrone

Questo teorema, di fondamentale importanza a garanzia dell'applicabilità di un percettrone alla risoluzione dei problemi linearmente separabili, ha il seguente enunciato [Bish95]:

> *Per qualsiasi problema di classificazione linearmente separabile, applicando la regola di addestramento di Rosenblatt (cfr. paragrafo 3.2) si trova certamente una soluzione con un numero* finito *di tentativi.*

La dimostrazione sarà data nel caso di un percettrone con un numero imprecisato di neuroni A (input) ed un solo neurone R (output). In questo caso, si noti, quella che è stata definita generalmente la "matrice" dei pesi, in realtà è un vettore riga.

Considerando un insieme linearmente separabile rispetto al problema di classificazione in questione, dipendente da N parametri (i neuroni), esisterà certamente una matrice $\mathbf{w}'$ di pesi sinaptici per cui tutti i vettori di esempio vengono correttamente classificati, in modo che risulti, per ogni n:

$$\mathbf{w}' \cdot \mathbf{X}^n t^n > 0 \tag{C.1}$$

dove $\mathbf{X}^n$ rappresenta l'n-esimo vettore di attivazione A e t^n l'attivazione conseguente dell'unico neurone R. Il processo di aggiornamento dei pesi inizia con valori casuali di essi che, senza ledere la generalità, possono essere immaginati tutti nulli.

Applicando il criterio di Rosenblatt, ad ogni passaggio del procedimento di addestramento in cui il vettore proposto $\mathbf{X}^n$ viene interpretato erroneamente, il vettore $\mathbf{w}$ viene modificato come segue:

$$\mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} + \mathbf{X}^n t^n \tag{C.2}$$

ove τ funge da variabile temporale con un incremento discreto. Si supponga adesso che dopo un certo numero di tentativi, il vettore $\mathbf{X}^n$ venga classificato τ^n (ove n è un indice e non un esponente) volte in modo sbagliato; allora, il vettore $\mathbf{w}$, in questa fase del procedimento, sarà dato da:

$$\mathbf{w} = \sum_n \tau^n t^n \mathbf{X}^n \tag{C.3}$$

Moltiplicando scalarmente questo vettore con il vettore "ottimale" $\mathbf{w}'$, si ottiene:

$$\mathbf{w} \cdot \mathbf{w}' = \sum_n \tau^n t^n \mathbf{w}' \cdot \mathbf{X}^n \geq \tau \min(t^n \mathbf{w}' \cdot \mathbf{X}^n) \tag{C.4}$$

dove si è indicato con $\tau = \sum_n \tau^n$ la somma totale degli aggiornamenti dei pesi. La disuguaglianza è ottenuta sostituendo ogni vettore di addestramento con il più piccolo di essi.

Da quanto visto, consegue che il prodotto scalare della (C.4) è limitato inferiormente da una funzione che cresce linearmente con t.

Si consideri adesso la norma del vettore dei pesi:

$$\|\mathbf{w}^{\tau+1}\|^2 = \|\mathbf{w}^{\tau}\|^2 + \|\mathbf{X}^n\|^2 (t^n)^2 + 2\mathbf{w}^{\tau} \cdot \mathbf{X}^n t^n \leq \|\mathbf{w}^{\tau}\|^2 + \|\mathbf{X}^n\|^2 (t^n)^2 \tag{C.5}$$

Qui la disuguaglianza segue dal fatto che il vettore $\mathbf{X}^n$ deve essere stato classificato male, per cui il doppio prodotto in (C.5) deve essere negativo. Inoltre, dato che abbiamo un output bipolare, sarà certamente $(t^n)^2 = 1$ e $\|\mathbf{X}^n\|^2 \leq \|\mathbf{X}_{\max}\|^2$, dove $\mathbf{X}_{\max}$ è il vettore di esempio con la norma più grande. Di conseguenza, la norma di $\mathbf{w}$ soddisfa la seguente relazione:

$$\Delta\|\mathbf{w}\|^2 = \|\mathbf{w}^{\tau+1}\|^2 - \|\mathbf{w}^{\tau}\|^2 \leq \tau\|\mathbf{X}_{\max}\|^2 \tag{C.6}$$

il che significa che dopo τ aggiornamenti, risulta:

$$\|\mathbf{w}\|^2 \leq \tau\|\mathbf{X}_{\max}\|^2 \tag{C.7}$$

e quindi la norma di $\mathbf{w}$ non cresce più rapidamente di $\sqrt{\tau}$. Ricordando adesso la conclusione risultante dalla (C.4), si conclude che, se questa lunghezza è limitata inferiormente da una funzione lineare di τ e $\mathbf{w}'$ è fissato, per un valore di τ abbastanza grande i due risultati (C.4) e (C.7) divengono incompatibili. Da questo si conclude che non può succedere che τ cresca indefinitamente, quindi il processo deve convergere dopo un numero finito di passaggi.

Appendice D

Teorema dell'energia nelle reti di Hopfield discrete con aggiornamento sequenziale

Su questo teorema [Kot93] si basa la sicurezza che una rete di Hopfield sia in grado di fornire, dopo un numero finito di passaggi (epoche), una risposta, a prescindere dal fatto che sia accettabile o meno:

> *L'energia di una rete neurale di Hopfield ad attivazioni binarie, aggiornata con criterio seriale, in occasione dell'aggiornamento di ogni singola unità, diminuisce o rimane costante.*

Si supponga di avere un dato stato di attivazioni della rete, contraddistinto da un vettore $(X_1, \ldots, X_i, \ldots, X_n)$. Il significato di questo vettore è chiaro nel testo della tesi. Supposto di aver compiuto un aggiornamento dell'i-esima unità, tale vettore cambia in modo da diventare $(X_1, \ldots, Y_i, \ldots, X_n)$, modificando solo l'i-esima attivazione. Si calcoli ora:

$$\Delta E = E(X_1, \ldots, X_i, \ldots, X_n) - E(X_1, \ldots, Y_i, \ldots, X_n) \tag{D.1}$$

quello che si vuole dimostrare è che $\Delta E \leq 0$.

Sulla base della definizione dell'energia, tutti i termini che non contengono l'attivazione i-esima sono uguali nel primo e nel secondo addendo del secondo membro della (D.1), quindi, sommando, spariscono. Allora risulta chiaramente:

$$\Delta E = -(Y_i - X_i)\left(\sum_{j=1}^{N} W_{ij}X_j - S_j\right) = A(X_i - Y_i) \tag{D.2}$$

ove S_i è la soglia del neurone in questione. Adesso, basta ricordare che, se la sommatoria è maggiore di S_i, allora A è maggiore di zero e la (D.2) vale $A(X_i - 1)$, mentre nel caso opposto, A è minore di zero, e la (D.2) vale AX_i. D'altro canto, X_i può valere solo 0 od 1: se $X_i = 0$ si ha $-A$ quando la somma supera la soglia (con A positivo) e 0 altrimenti, e se $X_i = 1$, si ottiene 0 nel primo caso ed A (negativo) nel secondo. In ogni caso, la variazione di energia è minore o uguale a zero.

Appendice E

Manuale d'uso del software di ricostruzione neurale implementato per la tesi

Non si tratta di un programma molto complesso, n dotato di interfaccia grafica. È necessario, comunque, conoscere alcuni aspetti necessari per procedere all'impostazione dell'algoritmo di tracciamento. Il motivo fondamentale è che ragioni di tempo ed il fatto che la tesi non intende essere uno studio di informatica troppo specialistico, non hanno consentito di realizzare un software che consideri tutti i possibili errori di input o altre defaillances nell'impostazione dei parametri, per cui è importante impostare gli input nel modo corretto, onde evitare un blocco del sistema, il sovraccarico della memoria o la cancellazione di dati.

E.1 Struttura dei file di lettura e scrittura dati

Questi file, per comodità di interpretazione, vengono creati con un'estensione opportuna, che non è obbligatoria, in quanto sono tutti file di testo il cui nome viene stabilito dall'utente in uno dei passaggi da seguire per l'elaborazione, ma è comodo, e semplice da interpretare, uno schema in cui ad ogni tipo di file si assegni un'estensione precisa. Lo schema seguito è illustrato in tabella E.1 Come si è già introdotto nella tabella stessa, potrà capitare che un tipo di file sia denominato con la sua estensione, ad esempio il "file `.txt`" rappresenta un file di input, e così via.

`.txt`	File iniziale contenente i punti da interpretare
`.out`	Risultato dell'elaborazione neurale
`.res`	Risultato dell'analisi dei dati del file `.out`
`.all`	Parametri di controllo della rete
`.trk`	Struttura iniziale delle tracce

Tabella E.1: Tipi di file usati nell'analisi neurale implementata per il lavoro di tesi.

E.1.1 File di impostazioni (`.all`)

La creazione di questo file è molto semplice, una volta che siano note le parole chiave da usare. Unica cosa cui bisogna fare attenzione è il modo in cui il file viene creato. La sintassi corretta prevede diverse righe di testo, strutturate sempre con la sintassi seguente:

parola-chiave `valore`

senza altre aggiunte, altrimenti la subroutine di lettura non riesce ad interpretare il valore inserito, o legge un valore sbagliato. Quanto alla parola chiave, essa va inserita nel modo esatto in cui la si presenterà nello schema seguente, tenendo conto che la lettura è *case-sensitive*. In tabella E.2 sono riportate tutte le parole chiave, con la relativa spiegazione.

Parola chiave	Significato
`inhibit`	parametro al numeratore della connessione inibitoria
`excit`	parametro moltiplicativo che compare nella connessione eccitativa
`power`	esponente del coseno nella connessione eccitativa
`temperature`	temperatura MFT
`variation`	variazione media relativa sotto la quale la rete neurale si reputa stabile
`phi`	massima differenza nell'angolo azimutale ϕ tra gli estremi del segmento-neurone
`theta`	massima differenza nell'angolo polare θ tra gli estremi del segmento-neurone
`pad`	massima differenza di *pad-row* consentita tra gli hit che fungono da estremi del neurone
`input`	nome del file `.txt` in cui sono salvati gli hit da analizzare
`output`	nome del file `.out` in cui salvare le tracce trovate

Tabella E.2: Schema delle parole chiave da inserire nel file di configurazione.

Il programma è realizzato in modo tale da rendere necessario che tutti i parametri siano positivi. Il programma, allora, all'apertura del file, controlla che questa condizione sia soddisfatta e interrompe l'esecuzione se ciò non succede. Lo stesso risultato è prodotto da un file di parametri in cui non compaiano *tutti* i valori della tabella.

E.1.2 File di input principale (`.txt`)

Questo file deve constare semplicemente di tante righe quanti sono i punti, contenenti, nell'ordine, i seguenti valori:

1. x, y e z del punto;
2. tre indici di ID di traccia che indicano il numero di traccia di quegli hit che hanno contribuito alla determinazione di quel dato punto[1];

[1]Questo elemento è previsto perché il simulatore, vista la densità di tracce, suppone che tre tracce possono avere tre rispettivi punti tanto vicini che la TPC ne vede, in realtà, uno solo, ma visto che il generatore tiene memoria dei punti di ciascuna traccia, allora "ricorda" tutte e tre le tracce che contengono quel punto. Naturalmente, ciò on è obbligatorio e magari, in una versione definitiva, in cui si lavori sui punti sperimentali, tale accorgimento decadrà.

3. il *pad-row* del punto;

4. le tre componenti del vettore impulso iniziale della particella;

5. il modulo dell'impulso trasverso;

Di questi valori, quelli ai punti 1, 4 e 5 sono numeri reali, mentre quelli ai punti 2 e 3 sono interi. L'unica operazione da fare, che è fattibile anche direttamente, è aggiungere una riga all'inizio del file, in cui scrivere il numero di punti in memoria. Ciò è stato scelto per questioni di rapidità nella raccolta dei dati: il programma, infatti, legge questo numero e dimensiona automaticamente l'array `Hit` di cui si è detto al capitolo 5, e poi procede all'acquisizione dei dati. Importante è che i diversi valori siano scritti in formato testo e separati da spazi, altrimenti il programma non riesce a leggere alcuni valori e posiziona nei diversi campi di ogni struttura `Point` creata i valori sbagliati.

E.1.3 File di struttura iniziale delle tracce simulate (`.trk`)

Questo file viene elaborato a partire dal precedente, per avere una visione completa di ogni traccia trovabile simulata, in modo da stabilire i parametri geometrici salienti di ogni traccia nel suo complesso e di ogni segmento tra due suoi hit consecutivi.

Per ogni traccia, in questo file si trovano:

- una riga contenente il suo ID ed il numero degli hit;

- una serie di righe in cui compaiono le componenti dell'impulso, l'impulso trasverso e gli angoli al vertice;

- una serie di righe in cui, per ogni pad-row della TPC, si riporta se è stato colpito; nel caso in cui ciò fosse successo, vengono riportate le coordinate dell'hit prodotto, eventuali ID di tracce con cui esso è condiviso (cfr. nota a pie' di pagina) e le differenze in angolo polare ed azimutale con l'hit della stessa traccia che lo precede nell'ordine temporale.

Questo file è servito a due scopi: uno è stato quello di valutare le prestazioni della rete in rapporto ai risultati della simulazione, il che è l'ultimo passaggio dell'analisi prodotta. L'altro scopo è stato quello di stimare correttamente, almeno a livello di analisi preliminare, i valori corretti da assegnare come parametri geometrici di cut, nella formazione dei neuroni (differenza di pad-row, e angoli ϕ e θ), in modo da poter avere una certa confidenza nel fatto che venissero creati almeno la maggior parte dei neuroni "giusti", ossia quelli che connettevano le coppie di hit consecutivi della stessa traccia, il che è necessario se si vuole che, alla fine, la traccia stessa venga trovata.

E.1.4 File di output della rete (`.out`)

Il file di output della rete viene prodotto in modo semplice. Quando la rete neurale ha generato il proprio output, ogni traccia che viene trovata, viene percorsa lungo il verso del vettore impulso, e vengono salvate le coordinate di ogni suo punto. Il file ha la struttura seguente:

- una riga contenente il numero di punti associati alla traccia (intero);

- tante righe (quanto è il numero di cui al punto precedente) contenenti le coordinate dei punti associati.

Alla fine dell'inserimento, viene posto un valore 0 predefinito, che serve all'interprete dei risultati per stabilire quando il file è terminato, dato che aprendo un file in formato testo, come è per quelli in uso, il controllo disponibile non riesce subito a "rendersi conto" di aver raggiunto la fine del file, e per questo registra valori senza senso, che potrebbero essere interpretati male e produrre risultati privi di significato.

E.1.5 File di analisi (`.res`)

Questo file viene prodotto dal programma che interpreta i risultati della rete. Tale programma restituisce, per ogni traccia, se essa risulta "trovata" in base al criterio enunciato nel capitolo 5, ossia se contiene almeno 32 punti, i valori dei parametri fisici di interesse analitico, che salva nel file. Quest'ultimo file prodotto ha uno schema che prevede una riga di testo per ogni traccia, contenente i seguenti valori:

- il track ID della traccia inizialmente simulata;
- il giudizio sulla traccia, che può essere `GOOD` per una traccia buona, o `FAKE` per una traccia non buona;
- i valori relativi al fit della traccia, che sono, nell'ordine in cui vengono scritti:
  - impulso trasverso;
  - angolo λ;
  - angolo ϕ;
  - parametro d'impatto longitudinale (l'unico non nullo se si effettua il best-fit della traccia con il vertex constraint)

Questo è il file che poi viene letto nella sessione di `ROOT` per valutare le statistiche di qualità della risposta della rete neurale sui dati prodotti.

E.2 Sintassi dei comandi di elaborazione

Tutti i programmi realizzati funzionano come comandi da inserire dal prompt di sistema. A questo proposito, è il caso di specificare che tutto il software ha girato su un PC in cui è stato installato Linux RedHat versione 6.2. Ogni programma richiede, per la lettura dei dati e la scrittura dei risultati, i nomi dei file in cui leggere e scrivere, che appartengono sempre ad una delle classi di file in tabella E.1. Per questo, esponendo la sintassi dei comandi definiti per far procedere il computer alle diverse fasi dell'analisi, si userà sempre una notazione del tipo:

comando `file1.ext [file2.ext] ...`

in cui *comando* è il nome del programma e i nomi seguenti sono quelli dei file che servono al suo funzionamento, di cui viene specificata l'estensione per capire di che tipo debbono essere, tra quelli in tabella E.1. Il primo passo è interpretare il file `.txt` per produrre un file `.out`, e lo si fa con il comando

```
net file1.all
```

in cui `net` è il programma di interpretazione. Viene specificato solo il nome di un file, in quanto questo file contiene quelli di input e di output (si è scelta questa struttura per semplificare l'immissione dei dati). Si prevede che il file `.out` venga creato (altrimenti il programma lo sovrascrive), mentre quello di input deve esistere già, altrimenti il programma segnala di non averlo trovato e si arresta.

Quando il programma è in esecuzione, compare una lista dei valori assegnati a tutti i parametri di controllo, in modo che un eventuale errore possa essere individuato e corretto prima di rischiare un blocco del sistema o errori di interpretazione. In fondo a quest'elenco, appare la richiesta di conferma, che, se riceve risposta negativa, interrompe l'esecuzione, per consentire di procedere con le correzioni. Se, leggendo il file dei parametri, qualcuno dovesse avere valori non consentiti, o mancare, il programma termina immediatamente.

Fatto questo, il programma inizia la sua esecuzione vera e propria, inviando alcuni messaggi che comunicano la fase della stessa cui è giunto. Alla fine dell'elaborazione, termina automaticamente e torna alla riga di comando del sistema operativo.

A questo punto, si usa il secondo programma, che richiede la seguente sintassi:

```
net_eval file1.out file2.res file3.txt
```

in cui `net_eval` è il programma di analisi. Esso chiede, quindi, un output di rete, un file di risultati in cui scrivere ed il file da cui la rete ha tratto i dati che ha analizzato. Quest'ultimo serve per calcolare gli scarti delle grandezze fittate rispetto a quelle iniziali, il che serve per i confronti di cui si è visualizzato il risultato nella seconda serie di immagini alla fine del capitolo 5. Anche qui non viene chiesto se è il caso di sovrascrivere un eventuale file `.res` già esistente.

Prima di elaborare l'analisi delle tracce trovate in modo grafico, è necessario utilizzare il programma `init file1.txt file2.trk`, per creare il file che compare a secondo argomento, e che è anch'esso necessario per la valutazione dei risultati della rete.

Per visualizzare le statistiche, bisogna far partire il framework `ROOT`, e dal suo prompt, eseguire una macro, che è chiamata

```
Graph.C(file1.res, file2.trk, eff, data, all)
```

Questa macro ha la possibilità di disegnare tre tipi di istogramma:

1. un istogramma che disegna il numero effettivo di tracce simulate, trovate `GOOD` e trovate `FAKE`, per un confronto quantitativo, disposte in intervalli di impulso trasverso;

2. un istogramma di efficienza, in cui viene graficato il numero di tracce trovate `GOOD` e `FAKE`, in rapporto al numero di tracce effettivamente simulate in partenza: anche questo istogramma è suddiviso in intervalli di impulso trasverso;

3. una quaterna di istogrammi in cui sono graficati il parametro d'impatto longitudinale (ottenuto dal fit), e la risoluzione negli angoli λ e ϕ dell'impulso al vertice, nonché la risoluzione dello stesso impulso trasverso ottenuto dalla procedura di best-fit.

Tutti questi dati vengono prodotti nell'elaborazione del programma `net_eval`. Gli argomenti della macro sono cinque: i primi due sono i nome di un file `.res` e di uno `.trk`

prodotti precedentemente, per i dati trovati e quelli di partenza con cui confrontarli; gli altri tre parametri sono variabili booleane che indicano alla macro se deve disegnare ciascuno dei tre istogrammi di cui sopra. Il parametro `eff` determina il disegno dell'istogramma al punto 2, il parametro `all` quello al punto 1 e il parametro `data` quello al punto 3.

Gli output di questa macro sono dello stesso tipo di quelli che sono stati presentati nel capitolo 5 per mostrare i risultati della rete. Si tratta di una semplice constatazione visuale, che serve a illustrare la bontà dei risultati, e può essere prodotta semplicemente dalla finestra che viene creata, salvando in formato `GIF`, usando l'apposito comando dal menu `File` che si trova nella finestra grafica stessa.

Bibliografia

[Testi]

[Bish95] *C. M. Bishop.* **Neural networks for pattern recognition**. Oxford University Press (1995).

[Free91] *J. A. Freeman, D. M. Skapura.* **Neural networks: algorithms, application and programming Techniques**. Addison-Wesley (1991)

[Hua97] *K. Huang.* **Meccanica statistica**. Zanichelli (1997)

[Pat87] *R. K. Pathria.* **Statistical Mechanics**. Pergamon Press (1987)

[Rin97] *G. Rindi, E. Manni.* **Fisiologia umana**. U.T.E.T. (1997)

[Wong94] *C. Wong.* **Introduction to high-energy heavy-ion collisions**. World Scientific (1994)

[Articoli da riviste scientifiche]

[Den88] *B. Denby.* Comp. Phys. Comm. **49** (1988) 429

[Den00] *B. Denby.* Comunicazione personale via email.

[Sti91] *G. Stimpfl-Abele, L. Garrido.* Comp. Phys. Comm. **64** (1991) 46

[Pet89] *C. Peterson.* Nucl. Instr. and Meth. in Phys. Res. **A279** (1989) 537-545

[Hum90] *B. Humpert.* Comp. Phys. Comm. **58** (1990) 223-256

[Bol95] *G. Bologna.* Il Nuovo Saggiatore **11** (1995) 50-71

[Pet88] *C. Peterson, J. R. Anderson.* Complex Systems **2** (1988) 59-89

[Kot93] *S. C. Kothari, K. Oh.* Advances in Computers **37** (1993) 119-166

[Kir83] *S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi.* Science **220** (1983) 671-680

[Sal88] *P. Salamon, J. D. Nulton, J. R. Harland, J. Pedersen, G. Ruppeiner, L. Liao.* Comp. Phys. Comm. **49** (1988) 423-428

[Note CERN e ALICE]

[ALICE-FTF] *A. Badalà, R. Barbera, M. Gulino, A. Palmeri, G. S. Pappalardo, F. Riggi, S. Vanadia.* ALICE Internal Note (02/2000), [in stampa]

[ALICE-TP] *ALICE Collaboration.* **Technical Proposal**. CERN/LHCC 95-71.

[TDR-TPC] *ALICE Collaboration.* **Time Projection Chamber, Technical Design Report**. CERN/LHCC 2000-001.

[AliRoot] Tutti i dettagli su `AliROOT` possono essere trovati nel sito web ufficiale dell'ALICE Off-line Project all'URL: `http://AliSoft.cern.ch/offline/`.

[ROOT] *R. Brun, F. Rademakers.* **Proceedings of the AIHENP '96 Workshop**. Lausanne, (1996); e *R. Brun, F. Rademakers.* Nucl. Instrum. Methods Phys. Res. **A389** (1997) 81. La pi completa ed aggiornata sorgente di informazioni su ROOT il suo sito web ufficiale: `http://root.cern.ch`.

[GEANT3] *CERN Application Software Group.* **GEANT: Detector Description and Simulation Tool**. (CERN, Geneva, 1993); CERN Program Library Long Write-ups W5013.

[KALMAN] *M. Regler, R. Früwirth.* **Reconstruction of charged tracks**. in Proceedings of the Advanced Study Institute on Techniques. e **Concepts in High Energy Physics**, St. Croix, 1988, Vol. 5 (Plenum, 1990), pp. 407-499.

[FTF] *P. Yepes.* Nucl. Instrum. Methods Phys. Res. **A380** (1996) 582

Indice