



UNIVERSITÀ DEGLI STUDI DI MILANO

FACOLTÀ DI SCIENZE E TECNOLOGIE

Corso di Laurea Triennale in Fisica

Analisi del Perceptron e della sua espressività nella classificazione di dati strutturati

Relatore

Prof. Marco Cosentino Lagomarsino

Correlatore

Prof. Marco Gherardi

Candidato

Andrea Lazzari

Matricola: 885250

Anno Accademico 2019-2020

Indice

1	Introduzione	1
1.1	Il machine learning e le sue basi teoriche	1
1.1.1	Cenni Storici e legami tra fisica e deep learning	2
1.2	Reti neurali artificiali: il sistema nervoso come modello	3
1.2.1	Tipi di architetture	6
1.2.2	Caratteristiche	7
1.2.3	Apprendimento Supervisionato	8
1.3	Classificazione lineare: il Perceptron	9
1.4	Classificazione di dati strutturati	12
1.4.1	Approssimazione di mean field	14
2	Risultati Teorici: probabilità di apprendimento ed espressività	15
2.1	Probabilità di separazione per dati puntiformi	15
2.1.1	Ipotesi della General Position	15
2.1.2	Probabilità di apprendimento: dicotomie totali e linearmente separabili	17
2.2	Cover e il Function Counting Theorem	19
2.2.1	Un parametro chiave: la capacità	28
2.3	Probabilità di separazione per doppietti monodispersi	33
2.3.1	Solo una parte delle dicotomie è realizzabile: la grandezza Ψ .	33
2.3.2	Dicotomie linearmente realizzabili su doppietti monodispersi .	35
2.3.3	Come cambia la capacità per dati strutturati	36
3	La programmazione lineare	41
3.1	Modello di ottimizzazione lineare	42
3.1.1	Dualità dei problemi lineari	43
3.2	Problemi continui: Algoritmo del simplesso	45
3.3	Problemi interi: Algoritmo Branch and Bound	48

4 Simulazioni numeriche e risultati ottenuti	51
4.1 Misura numerica della probabilità di separazione con metodi di linear programming	51
4.1.1 Funzionamento dell'algoritmo	52
4.1.2 Risultati per input non strutturati	58
4.2 Algoritmo per input a doppietti	62
4.2.1 Risultati per input a doppietti	66
5 Conclusioni	71
6 Appendice	75
6.1 Calcolo di $\Psi_2(\rho)$	75
6.2 Teoria di Cover per politopi generici (k -upletti)	76
6.3 Esempio di applicazione del metodo del simplesso	79
6.4 Esempio di applicazione dell'algoritmo Branch and Bound	82
Bibliografia	84

Capitolo 1

Introduzione

" Il cambiamento è il risultato finale del vero apprendimento: ogni volta che impariamo qualcosa di nuovo, noi stessi diventiamo qualcosa di nuovo "

Proprio come nella frase dello scrittore statunitense Leo Buscaglia, l'apprendimento è un passo fondamentale della vita umana: dalle azioni più basilari, quali camminare o riconoscere oggetti, ai problemi più complessi, infatti, l'uomo impara attraverso l'osservazione, l'imitazione di comportamenti, la lettura, lo studio, l'interazione, la comunicazione e tutte le altre attività che ci caratterizzano. Da queste non facciamo che estrarre informazioni che, con l'esperienza, diventano regole e modelli da utilizzare nella vita. L'apprendimento non è lineare, bensì un processo iterativo che all'aumentare delle informazioni raccolte, migliora la nostra conoscenza.

L'obiettivo del Machine Learning è proprio quello di *ricreare* e utilizzare questo processo: dato un algoritmo di apprendimento, quest'ultimo ne analizza dati ed esempi, per estrarne le regole che li descrivono.

1.1 Il machine learning e le sue basi teoriche

Trovandosi davanti a problemi sempre più difficili, l'uomo ha dovuto reinventarsi e sviluppare nuove tecniche risolutive; nel progredire, infatti, lo studio scientifico si è servito della tecnologia per comprendere sempre meglio i meccanismi e le leggi presenti in natura. Nel cercare di comprendere più a fondo il sistema nervoso, o più in generale l'intelligenza umana, si è arrivati alla costruzione delle cosiddette

Intelligenze Artificiali (il cui acronimo AI), ovvero alla riproduzione informatica (e non solo) della macchina più complessa in natura: l’Uomo.

Nelle molteplici applicazioni dell’Intelligenza Artificiale alla base di tantissime tecnologie moderne, quali riconoscimento del linguaggio e della grafia, classificazione di immagini, guida autonoma, l’AI è innanzitutto programmata ed istruita a risolvere un problema assegnato e dopodichè impiegata ad affrontare problemi simili. Prendendo un esempio specifico, si può considerare un programma informatico che analizzi un certo insieme di fotografie e classifichi quali raffigurano cani e quali gatti; in questo caso occorre innanzitutto programmare l’AI in modo che conosca i parametri con cui classificare le foto e, successivamente, istruirla su quali valori dei parametri vada associato un cane e su quali un gatto.

Il processo per cui una AI impara a risolvere un dato obiettivo, un dato *task*, prende il nome di Machine Learning. L’idea è sfruttare le capacità computazionali di un computer in una struttura ispirata al sistema nervoso umano: per una persona, operazioni come riconoscere un volto in una foto, o estrapolare un testo da un’immagine sono infatti svolte in maniera estremamente naturale. Su queste dinamiche si basa l’argomento centrale di questo elaborato: le reti neurali artificiali.

1.1.1 Cenni Storici e legami tra fisica e deep learning

Nei primi anni ’40 del XX secolo, Warren McCulloch e Walter Pitts furono i primi a descrivere un modello che collegava delle unità elementari, i *neuroni artificiali*, basandosi sulla struttura del cervello umano. Anche se nel 1949 Hebb ipotizzò la possibilità di istruire le macchine con un apprendimento che emulava quello alla base dell’intelligenza umana, il primo prototipo vero e proprio di rete neurale fu proposto qualche anno dopo, nel 1958, quando Frank Rosenblatt, uno psicologo statunitense diede vita al Perceptron. Costituito da uno strato di nodi di input e da un solo nodo di output, esso era capace di riconoscere forme e classificarle in due gruppi separati.

Questa rete neurale sarà il fulcro dell’elaborato e se ne tratterà il comportamento in dipendenza ai dati forniti in input.

Da qui in poi le tecnologie legate al machine learning non si sono più fermate e hanno coinvolto tantissimi ambiti, nel 1997 IBM Deep Blue fu il primo calcolatore in grado di sconfiggere il campione del mondo di scacchi e nell’ultimo decennio i metodi di apprendimento automatico hanno registrato numerosi successi per quanto riguarda il riconoscimento e la elaborazione di immagini, in medicina e nel mondo

1.2. RETI NEURALI ARTIFICIALI: IL SISTEMA NERVOSO COME MODELLO

della finanza, dove con esse si sono effettuate previsioni sull'andamento dei mercati e analisi di rischio.

L'insieme delle tecniche moderne e degli algoritmi basati sulle reti neurali artificiali, prende il nome di *Deep Learning*. Questo campo di ricerca è particolarmente collegato alla fisica, in quanto le reti neurali sono strumenti utilizzati per risolvere problemi molto complessi, dalla fisica delle particelle alla meccanica quantistica e, come si vedrà in seguito, i loro comportamenti hanno forti legami con la meccanica statistica. Nonostante i grandi progressi degli studi più recenti, rimangono ancora molte domande per quanto riguarda le *deep neural networks*: sono difficili da interpretare, manca una solida comprensione di quando le risposte fornite siano soddisfacenti e, soprattutto, sono difficili da adattare a nuovi problemi.

Per comprendere meglio, si potrebbe confrontare lo stato della teoria del Deep Learning alla teoria della luce e della materia all'inizio del ventesimo secolo. Molti furono i risultati ottenuti da esperimenti (come ad esempio l'effetto fotoelettrico) che non potevano essere spiegati da una teoria matematicamente rigorosa: la meccanica quantistica infatti, doveva ancora essere sviluppata.

Proprio come accade per il machine learning, la teoria esistente ha ancora delle mancanze e non è in grado di descrivere con il giusto rigore matematico i molti contributi esistenti dal punto di vista applicativo. Inizialmente i primi modelli che si sono formulati con approcci di meccanica statistica dei sistemi disordinati, consideravano semplici connessioni neurali e con dati in input senza alcuna struttura; tuttavia, con i progressi in ambito di deep learning, l'approccio della fisica alla ricerca in questi ambiti ha apportato contributi interessanti e ha sviluppato nuove teorie per spiegare i comportamenti di reti neurali sempre più complesse.

1.2 Reti neurali artificiali: il sistema nervoso come modello

Da sempre la natura, nelle sue varie espressioni, è sempre stato motivo di ispirazione, offrendo forme da rappresentare, fenomeni da modellizzare e metodi da imitare. Il concetto di *rete neurale* non appartiene solamente all'informatica, bensì è particolarmente generale e fonda le sue radici nella biologia, dove è normalmente associato alla struttura del "sistema nervoso". Il "mattone elementare" costituente

1.2. RETI NEURALI ARTIFICIALI: IL SISTEMA NERVOSO COME MODELLO

le strutture cerebrali è una cellula denominata *neurone*, che è sede di processi elettrochimici responsabili della generazione di campi elettromagnetici.

Nel cervello umano sono presenti oltre 100 miliardi di neuroni, ciascuno interconnesso circa ad altri 10.000 attraverso le *sinapsi*, strutture per mezzo delle quali un'informazione viene trasmessa e processata.

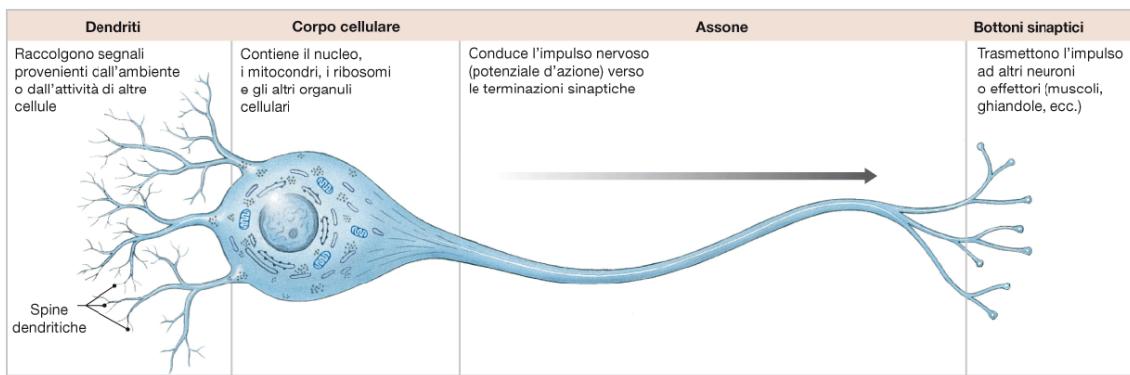


Figura 1.1: Composizione schematica di un neurone

Per capirne il funzionamento si può considerare il meccanismo per cui l'uomo è in grado di avere percezioni sensoriali, ad esempio il tatto. La nostra pelle è costellata di recettori, alcuni sensibili allo scambio di calore (ci permettono di sentire il caldo e il freddo), alcuni alle irregolarità delle superfici (per distinguere il liscio dal ruvido), altri alla pressione (che danno sensazione di morbidezza o rigidità) ... etc. Ognuno di questi recettori trasforma un segnale proveniente dall'esterno in un segnale elettrico, il quale, dopo essere stato inoltrato al neurone più vicino, verrà diffuso attraverso la rete nervosa fino al raggiungimento del cervello, dove sarà processato e darà forma alla percezione vera e propria.

È importante però capire come avviene questo trasporto e il *processing* dell'informazione. Ogni recettore emette un segnale elettrico di intensità proporzionale allo stimolo che riceve dall'esterno, verso i neuroni a cui è interconnesso. Attraverso questa interconnessione un neurone può attivarsi (o disattivarsi) a seconda del fatto che il segnale che gli giunge abbia un'intensità maggiore (o minore) di una certa soglia. L'intensità del segnale trasmesso verso i neuroni successivi fino al cervello, sarà quindi modulata in modo dipendente sia dal livello di attivazione di ogni neurone della catena, sia dai tipi di interconnessioni costituenti la rete. Gli stati di attivazione neuronali non sono necessariamente due (attivo o disattivo), bensì normalmente si ha anche la presenza di valori intermedi, potenzialmente un continuo di valori.

1.2. RETI NEURALI ARTIFICIALI: IL SISTEMA NERVOSO COME MODELLO

Una volta che gli impulsi dei vari stimoli esterni raggiungono il cervello, quest'ultimo rielabora le varie informazioni sulla base dell'intensità e dai recettori e neuroni che sono stati coinvolti. Per esempio, se si immerge un dito in acqua calda il cervello riceverà un segnale debole da parte dei neuroni maggiormente influenzati dai recettori della pressione, mentre riceverà un segnale intenso dai neuroni influenzati dai recettori del calore. Inoltre, più complessa è la rete neurale, maggiori sono le possibili sfumature di percezioni che il cervello è in grado di interpretare. Le interconnessioni del sistema nervoso umano in aggiunta sono multiple, ossia ogni neurone è collegato a diversi neuroni che lo circondano, pertanto l'attivarsi (o il disattivarsi) di un neurone influenza a cascata un gran numero di cellule nervose.

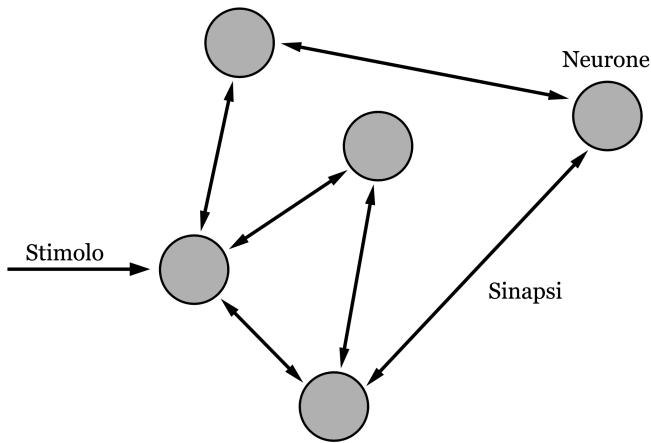


Figura 1.2: Schema semplificato di una rete neurale biologica

Compreso il modello biologico di riferimento, il passo successivo è quello di adattarlo ad una circostanza più specifica: le reti neurali artificiali in ambito informatico. Una rete neurale informatica è organizzata in neuroni interconnessi che si influenzano vicendevolmente attraverso un potenziale di interazione, con la differenza che in questa situazione i costituenti sono oggetti virtuali e non materia biologica. Esistono vari tipi di reti neurali basati su diverse architetture che lavorano con modalità differenti per arrivare ad implementare un determinato compito cognitivo.

1.2.1 Tipi di architetture

Come visto precedentemente, l'apprendimento di uno stimolo esterno da parte del sistema nervoso umano avviene in più fasi: la traduzione dei recettori in impulso elettrico, il trasporto dell'impulso da parte dei neuroni periferici e infine l'elaborazione da parte del cervello. Ogni gruppo di neuroni è specializzato, ovvero è adibito ad eseguire una singola di queste operazioni, proprio come accade nelle reti artificiali, dove questi "gruppi" sono chiamati *layer* e rappresentano gli strati di apprendimento che si susseguono dall'input iniziale (che nel caso biologico è l'oggetto esterno) fino all'output finale (ovvero la risposta che il sistema nervoso dà allo stimolo).

Sono sempre necessari almeno due layer, quello di ingresso e quello di uscita, i quali formano un'architettura di tipo *single-layer*. Nel caso in cui siano presenti uno o più strati intermedi (detti *hidden layers*), l'architettura sarà di tipo *multi-layer*. Una struttura *single-layer* è molto più rapida, facile da realizzare e se ne può studiare più agevolmente il comportamento; al contrario, una rete *multi-layer* sarà in grado di svolgere compiti più complessi a spese però, di una maggior complessità realizzativa.

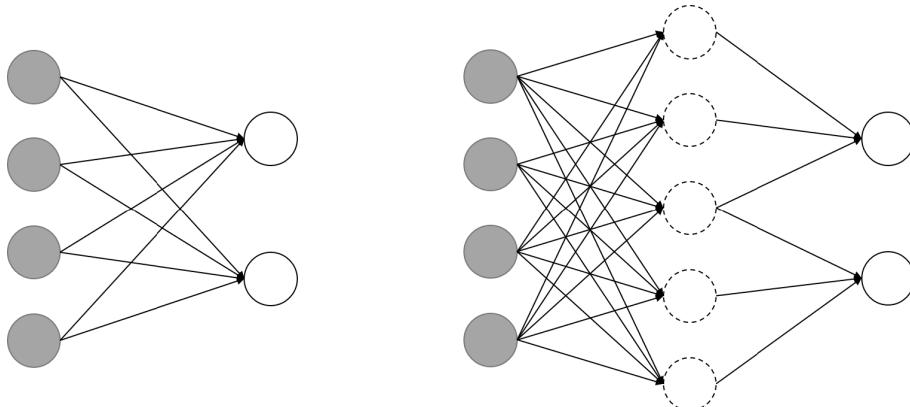


Figura 1.3: A sinistra: rappresentazione schematica di un'architettura single-layer; a destra: schema di una rete multi-layer. In grigio viene rappresentato il layer di input, in bianco con bordo continuo il layer di output e in bianco con bordo tratteggiato lo hidden layer

Un altro elemento caratterizzante una rete neurale è la tipologia delle interconnessioni tra i neuroni. Ne esistono di diversi tipi, due però sono particolarmente importanti: interconnessioni con *feedback* e interconnessioni *feedforward*. Nelle reti di quest'ultimo tipo, un neurone appartenente ad un certo layer può essere influenzato soltanto da neuroni situati nei layer "precedenti" ad esso, ovvero nei layer corrispondenti ad uno strato di apprendimento inferiore,

1.2. RETI NEURALI ARTIFICIALI: IL SISTEMA NERVOSO COME MODELLO

situati più vicini al livello di input. Al contrario, nelle reti con meccanismi a *feedback*, ogni neurone può essere influenzato da neuroni appartenenti a qualsiasi layer (potenzialmente anche da se stesso).

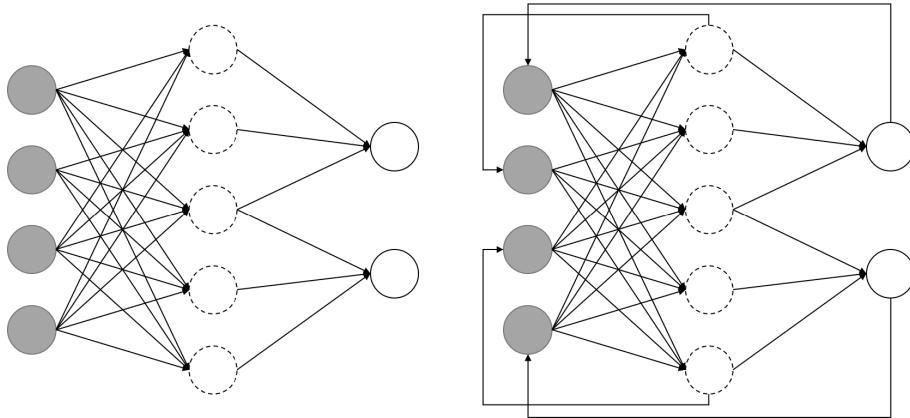


Figura 1.4: Schema di due reti neurali *double-layer* a sinistra con interconnessioni *feedforward* e a destra con meccanismo *feedback*

L’architettura della rete neurale, così come il tipo di interconnessioni, sono inevitabilmente dipendenti dal problema che si vuole risolvere: compiti più complessi necessiteranno di reti neurali più strutturate e nel caso in cui il problema che si sta affrontando presenti ad esempio la necessità di retroazione, risulterà essenziale un meccanismo di *feedback*. Il sistema nervoso dell’uomo è una rete neurale con un meccanismo di feedback caratterizzato da un particolare *pattern* di interconnessioni. Alcuni studi inoltre evidenziano come la struttura cerebrale e le sinapsi siano influenzate dalla vita degli individui, dalle esperienze e dall’apprendimento di compiti specifici.

1.2.2 Caratteristiche

Per entrare più nel dettaglio riguardo la struttura di una rete neurale se ne descrivono le grandezze e le caratteristiche dei vari componenti. Ad esempio una rete neurale feedforward a single-layer può essere paragonata ad un sistema nervoso costituito solo da recettori sensoriali (neuroni che ricevono gli stimoli di input) che, a seconda del potenziale di interazione, determinano immediatamente il label in risposta allo stimolo ricevuto (layer di output). Importante è descrivere i componenti fondamentali del sistema, ossia i neuroni. Ognuno di essi rappresenta un criterio, una caratteristica secondo la quale viene effettuata una classificazione sui dati di input; ciò può essere rappresentato come una dimensione nello *spazio delle fasi*. Se una rete

neurale dispone di n neuroni, allora sarà possibile descrivere matematicamente la rete come uno spazio vettoriale n -dimensionale, dove ogni vettore di base rappresenta lo stato di attivazione del neurone corrispondente. Per rendere più chiaro questo concetto, si prenda ad esempio un sistema biologico dotato di recettori sensibili al calore, alla pressione e alle irregolarità superficiali: lo spazio delle fasi, in questo caso avrebbe *tre* dimensioni, ognuna delle quali correlata all'intensità dello stimolo del gruppo di recettori corrispondenti.

Un altro aspetto essenziale è quello degli input esterni: ogni input corrisponde ad un vettore nello spazio delle fasi (successivamente verrà indicato con il simbolo ξ) le cui proiezioni sui vari assi rappresentano il valore di attivazione di uno specifico neurone sottoposto a quell'input (biologicamente, ogni oggetto esterno che viene toccato attiverà differentemente i recettori tattili). In seguito il numero totale di input verrà indicato con il simbolo p .

Anche se l'obiettivo di una rete neurale sviluppata per il machine learning è quello di *apprendere*, ciò non sempre avviene con successo. Risulta quindi di grande interesse quantificare la probabilità per cui, data una certa architettura, la rete neurale porta a termine l'apprendimento. Tale probabilità dipende dal numero di neuroni che si usano, dal numero di input da classificare, ma anche dal tipo di interconnessioni e dal numero di output desiderati.

1.2.3 Apprendimento Supervisionato

Nei paragrafi precedenti si è parlato del meccanismo per cui il sistema nervoso umano può distinguere un certo stimolo da un altro e elaborare una risposta specifica, ad esempio nella distinzione tra il calore di quando si avvicina la mano ad un fuoco e la sensazione di morbidezza di un cuscino. Nel caso di reti neurali artificiali però, l'informazione di quale stimolo vada associato all'una o all'altra cosa è determinato dall'esperienza; la caratteristica più rilevante di una rete neurale ai fini del *machine learning* però è proprio la sua capacità di apprendere. Al contrario, in un sistema biologico la struttura è ben definita: il tipo di recettori, il numero di neuroni, la configurazione delle interconnessioni e molti altri "parametri" sono fissati e non possono essere alterati. Le percezioni sensoriali che il cervello riceve di un certo oggetto esterno, di conseguenza, sono sempre uguali e non possono essere modificate.

Nel processo di apprendimento *supervisionato* si attribuisce un "nome" (in seguito si utilizzerà il termine *label*) allo stimolo corrispondente. Metaforicamente si può considerare di dover insegnare ad un bambino (la rete neurale che deve

1.3. CLASSIFICAZIONE LINEARE: IL PERCEPTRON

apprendere) a riconoscere e attribuire il label "fuoco" quando qualcosa brucia dentro al camino, e il label "giocattolo" al modellino di una macchina da corsa (ovvero a realizzare una certa classificazione). Il bambino, oltre al diverso stimolo visivo, associerà lo stimolo tattile di forte calore al label "fuoco", mentre associerà uno stimolo legato alla pressione per quanto riguarda il label "giocattolo". Una volta fornite queste "coppie" di stimolo-label, il processo di apprendimento è concluso e, qualora la rete neurale ricevesse nuovamente uno stimolo comparabile ad uno dei due già conosciuti, saprà come classificarlo.

Bisogna però fare attenzione al fatto che, proprio come un bambino, la rete neurale tenderà a ricondurre ogni nuovo stimolo a quelli che ha già imparato. Ciò implica che ogni nuovo stimolo esterno con cui verrà a contatto, anche appartenente ad un'altra categoria, verrà classificato come "fuoco" o come "giocattolo", finché non venga fornito un label differente a cui associare lo stimolo. Legato a questo è il concetto di *espressività*, relativo alla quantità di classificazioni che una rete neurale è in grado di apprendere e realizzare, una sorta di *potenzialità*, efficienza della rete.

Nell'apprendimento supervisionato, quindi, si fornisce alla rete il cosiddetto *Training Set*, un insieme di input e di label corrispondenti $\{\xi^i, \sigma_i\}$; analizzandolo, la rete apprende la relazione che li lega imparando a generalizzare, ossia a calcolare nuove associazioni input-output processando input esterni al training set. Nella fase di analisi, per ogni input assegnato, la rete elabora secondo il proprio algoritmo una decisione a livello di output; dopodiché effettua un confronto con il label corretto fornito nel training set e trae alcune conclusioni che influenzano le valutazioni successive dell'addestramento. Come risultato si avrà una serie di parametri, chiamati *pesi* w , caratterizzanti l'algoritmo della rete neurale, calibrati per svolgere un dato *task*.

1.3 Classificazione lineare: il Perceptron

Il perceptron è una delle reti neurali più semplici che si possano realizzare: la sua struttura (figura (1.5)) prevede un interconnessione feedforward a single-layer caratterizzata da un output di tipo binario. Ciò significa che un perceptron raccoglie informazioni da un numero arbitrario di neuroni e, mediante il potenziale di interazione, assegna loro un label a scelta tra due.

Architettura introdotta per la prima volta nel 1958 dallo psicologo statunitense Frank Rosenblatt e che ha rappresentato un vero e proprio salto innovativo, in quanto

gettò delle vere e proprie basi per le intelligenze artificiali essenziali allo sviluppo di tantissime tecnologie moderne.

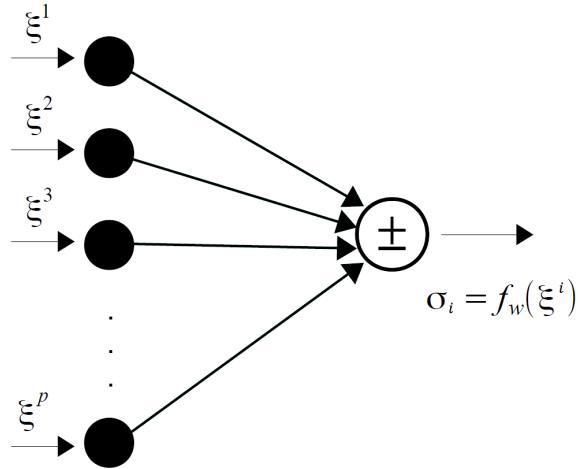


Figura 1.5: Schema della struttura di un Perceptron

Nonostante la sua semplicità il perceptron è ampiamente utilizzato nelle applicazioni pratiche, in particolare come elemento modulare per la realizzazione di architetture più complesse ed è quindi un interessante caso di studio. Le sue caratteristiche, inoltre, ne permettono uno studio del comportamento in forma analitica attraverso gli strumenti della meccanica statistica.

Per realizzare un output di tipo binario, è presente un solo neurone nel layer di output, il quale sarà quindi contraddistinto da due soli possibili stati di attivazione denominati $\sigma = \pm 1$. Fornito un certo input al perceptron, la funzione modello che "realizzerà" tale output sarà:

$$f_{\vec{w}}(\vec{\xi}^i) = \operatorname{sgn}\left(\sum_{i=1}^n w_i \cdot \xi^i\right) = \operatorname{sgn}(\vec{w} \cdot \vec{\xi}) \quad (1.3.1)$$

dove w_i rappresenta il valore del potenziale di interazione esercitato dall' i -esimo neurone di input sul neurone di output e ξ^i è lo stato di attivazione dell' i -esimo neurone indotto dall'input che si sta considerando; la funzione segno rende binario lo stato del neurone d'uscita.

Il vettore \vec{w} (detto vettore dei *pesi*) è il vero e proprio responsabile del processo di classificazione, vive nello spazio delle fasi e, in quanto vettore, identifica un iperpiano passante per l'origine che separa lo spazio in due ipervolumi, uno contenente \vec{w} e l'altro no. Proprio per questo fatto il perceptron è anche detto *classificatore dicotomico lineare*, ovvero esegue dicotomie lineari, cioè separazioni dei dati in due sottoinsiemi disgiunti attraverso un iperpiano.

La funzione *forward* che realizza questa classificazione dei dati è proprio la relazione (1.3.1); il prodotto scalare tra i due vettori infatti, nel caso in cui $\vec{\xi}$ sia contenuto nel semispazio contenente \vec{w} , sarà positivo e $\sigma = +1$, altrimenti si avrà $\sigma = -1$. Modificare \vec{w} corrisponde a variare l'inclinazione dell'iperpiano, ottenendo una diversa separazione dello spazio delle fasi e, di conseguenza, dei dati in esso contenuti. Nel caso in cui si abbia da classificare non un singolo input ma un insieme di punti $\{\xi^1, \dots, \xi^p\}$, occorre che il vettore \vec{w} soddisfi contemporaneamente tutte le classificazioni richieste per i vari input (label del training set).

In figura si può facilmente notare che non sempre è possibile che una rete neurale apprenda una specifica classificazione.

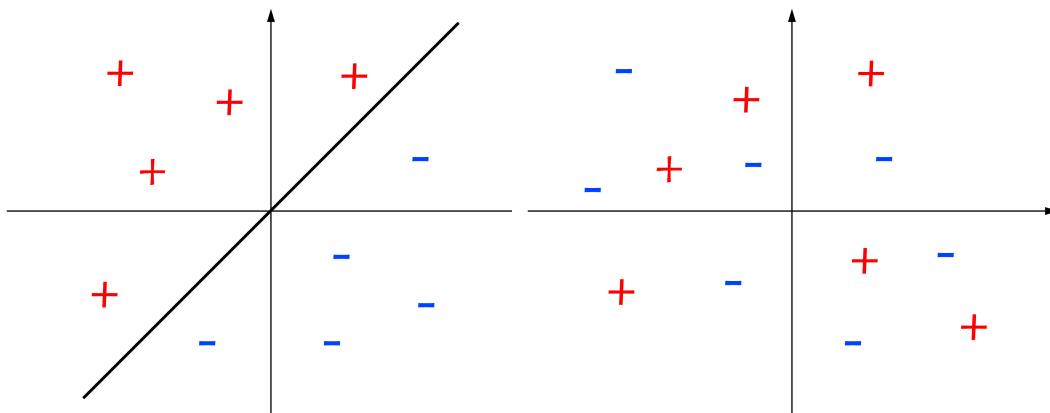


Figura 1.6: A sinistra una dicotomia linearmente separabile che risulterà apprendibile da un perceptron; a destra invece, una tale disposizione dei dati non porterà ad alcun apprendimento

Il fatto che un perceptron esegua separazioni lineari attraverso un iperpiano, implica che ogni vettore $\vec{\xi}$ può essere normalizzato sull'ipersfera di raggio unitario centrata nell'origine senza alcuna perdita di informazione. Infatti il trovarsi in un certo ipervolume rispetto al piano, è dovuto solamente alla *direzione* e non al modulo del vettore in ingresso $\vec{\xi}$. Questo fatto permetterà di vincolare gli input all'ipersfera unitaria.

Da questa descrizione del perceptron si evince subito che, nel caso di apprendimento possibile, i valori di \vec{w} che classificano correttamente gli input possono essere molteplici. Si definisce la cosiddetta *Loss Function* \mathcal{L} :

$$\mathcal{L}_{\vec{w}} = \sum_{i=1}^p \theta(-f_{\vec{w}}(\vec{\xi}^i) \cdot \sigma_i) \quad (1.3.2)$$

dove la funzione θ di Heaviside determina la presenza di due casi:

1. Se l'output perceptron $f_{\vec{w}}(\vec{\xi}^i)$ e il correlato label del training set σ_i hanno lo stesso segno (sono entrambi +1 o entrambi -1) si avrà:

$$\theta(-1) = 0 \quad (1.3.3)$$

e, conseguentemente, l'apporto alla Loss Function da parte di $\{\vec{\xi}^i, \sigma_i\}$ sarà nullo.

2. Se invece l'output del perceptron $f_{\vec{w}}(\vec{\xi}^i)$ e il label σ_i hanno segno opposto (sono uno +1 e l'altro -1, o viceversa) si avrà:

$$\theta(+1) = 1 \quad (1.3.4)$$

di conseguenza l'apporto alla Loss Function da parte di $\{\vec{\xi}^i, \sigma_i\}$ sarà diverso da zero.

Da questi fatti la Loss Function \mathcal{L} (secondo la definizione (1.3.2)) rappresenta il numero di errori di classificazione commessi dal perceptron durante la fase di training. L'obiettivo del machine learning è trovare il vettore dei pesi \vec{w} che caratterizzano $f_{\vec{w}}(\vec{\xi}^i)$, ossia quelli con i quali gli output della rete neurale corrispondano il più possibile ai label presenti nel training set. Questo è equivalente al minimizzare gli errori commessi dalla rete in fase di apprendimento: per tali valori di \vec{w} la Loss Function \mathcal{L} , sarà minima.

1.4 Classificazione di dati strutturati

Le reti neurali artificiali sono oggi ampiamente utilizzate in diversi ambiti, soprattutto grazie all'efficienza delle architetture che si sono realizzate. Nonostante questo sono ancora molte le domande al loro riguardo e, oltretutto, non esiste ancora una teoria matematica completa che descriva analiticamente il funzionamento e le caratteristiche delle *neural networks*.

Quando si considerano situazioni reali alcune delle ipotesi che stanno alla base di molti studi analitici risultano troppo restrittive o comunque identificano casi fin troppo "ideali", come ad esempio l'assunzione che i dati di input siano punti nello spazio delle fasi distribuiti casualmente sulla sfera unitaria. Solitamente, infatti, accade che input aventi caratteristiche simili si trovino più vicini tra loro, nello spazio delle fasi, rispetto a dati con delle diversità. Per esempio, in un set di fotografie che ritraggono cani e gatti, ogni foto sarà rappresentata da un certo punto nello spazio

1.4. CLASSIFICAZIONE DI DATI STRUTTURATI

delle fasi; la rete neurale classificherà i cani diversamente dai gatti, ma al contempo avrà la tendenza ad identificare in modo simile cani della stessa razza o fotografie scattate dalla stessa prospettiva, raggruppando i dati di input nello spazio delle fasi. Questo tipo di "struttura" nei dati viene chiamata struttura a *object manifold* e, nello spazio delle fasi, corrisponde ad una moltitudine di nuvole di punti.

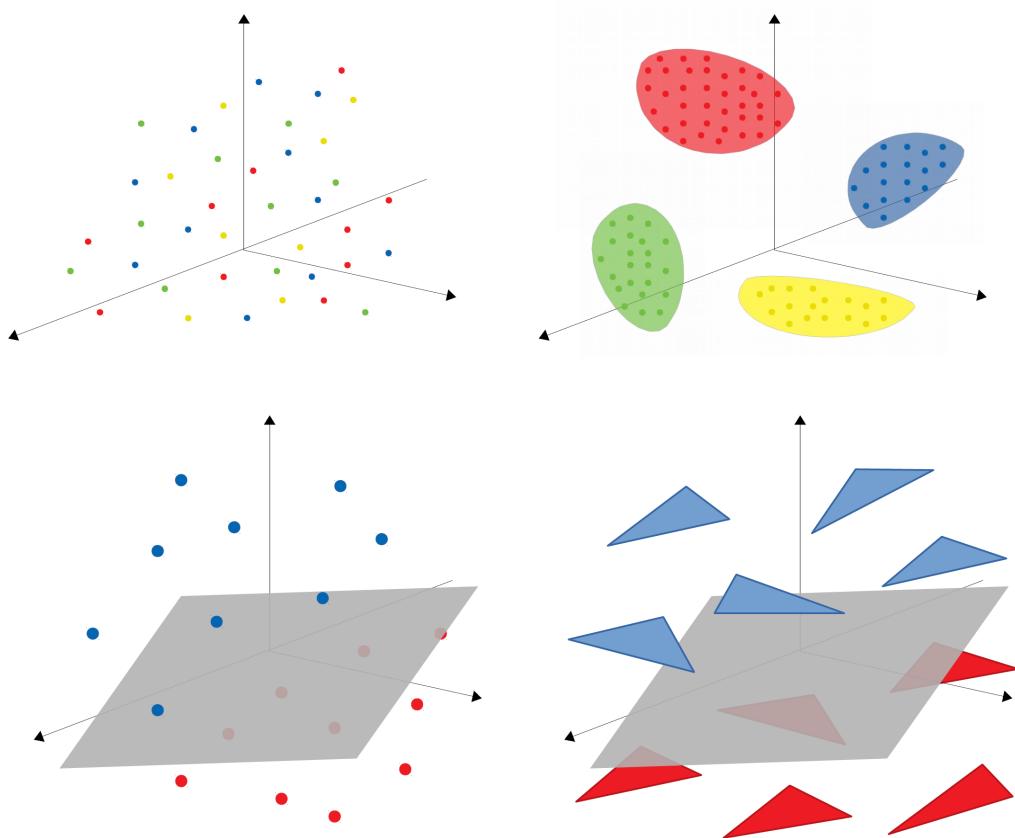


Figura 1.7: Dati senza struttura vs Dati strutturati

La struttura a manifold può influire positivamente sulla velocità di apprendimento di una rete neurale, perché la classificazione di punti uniformemente dispersi nello spazio è sicuramente più lenta di quella che si può effettuare su punti raggruppati in cluster. Di contro però, ciò rende più complessa una trattazione analitica dell'argomento in quanto subentrano nuovi parametri legati alle relazioni intrinseche come densità dei manifold o addirittura la presenza di intersezioni tra i gruppi di dati.

Molti studi si sono occupati di indagare l'influenza della presenza di una struttura dei dati sull'apprendimento delle reti neurali. In particolare in [1] è stata analizzata la probabilità che un perceptron apprenda una dicotomia nel caso in cui i dati siano costituiti da *politopi*, ovvero insiemi di punti aventi una struttura geometrica ben definita. Questi aggregati sono contraddistinti dal fatto che tutti i punti che li compongono si dispongono in modo da avere la stessa distanza reciproca. Nello studio [1] (e come si riprenderà in seguito) si sono raggiunti interessanti risultati analitici sotto un'ipotesi aggiuntiva: la *monodispersione*. In tale assunzione, gli input sono strutturati in politopi *monodispersi*, ovvero la distanza tra i punti dello stesso politopo è uguale per ogni politopo che costituisce il training set.

1.4.1 Approssimazione di mean field

L'ipotesi di monodispersione è abbastanza restrittiva in quanto solitamente, in situazioni più concrete, gli input possiedono strutture più diversificate. Si ha quindi la necessità di superare questo limite, indagando il dominio di validità dei risultati ottenuti in [1], soprattutto per quanto riguarda l'espressività del perceptron. In particolare, in questo elaborato ci si occuperà di studiare il comportamento di politopi composti da due punti, più comunemente chiamati *doppietti*.

Per "estendere" la validità dei risultati, si formula l'ipotesi che l'apprendimento di un perceptron rispetti un *mean field* rispetto alla polidispersione dei doppietti. Secondo questa approssimazione il numero di dicotomie realizzabili da un perceptron su input organizzati in doppietti polidispersi seguirà la stessa legge di apprendimento su doppietti monodispersi, a patto di considerare la distanza media dei punti all'interno dei doppietti. Ciò si raggiungerà con simulazioni numeriche che mettono a confronto il numero di dicotomie linearmente separabili $C(n, p)$ previsto dalla teoria analitica, con il numero di dicotomie effettivamente realizzato e appreso da un perceptron.

I risultati permetteranno una miglior comprensione della dinamica delle reti neurali nella classificazione di dati strutturati e soprattutto una maggior adattabilità del *back-ground* teorico a successive applicazioni.

Capitolo 2

Risultati Teorici: probabilità di apprendimento ed espressività

2.1 Probabilità di separazione per dati puntiformi

Da diversi anni in ambito di fisica e in particolare di fisica statistica le reti neurali sono un vero e proprio oggetto di studio. Esse sono infatti utilizzate in analisi basate sull'approccio standard della meccanica statistica: lo studio di sistemi disordinati nel limite termodinamico, ossia nell'ipotesi in cui il numero di gradi di libertà del sistema tenda all'infinito.

In questo campo di grande rilevanza sono i lavori di Gardner [2,3], riguardanti la quantità di informazioni che una rete neurale ideale può immagazzinare. In questo lavoro di tesi l'approccio scelto è differente, proprio come in [1,4] si cercano risultati analitici fissando al finito il numero di gradi di libertà: ciò permette una verifica dei dati sperimentali con simulazioni accurate e computazionalmente poco dispendiose.

2.1.1 Ipotesi della General Position

Il perceptron è una rete neurale che realizza dicotomie lineari.

Una dicotomia è linearmente realizzabile se l'insieme dei punti aventi lo stesso label può essere suddiviso nello spazio delle fasi in due sottoinsiemi disgiunti attraverso un iperpiano passante per l'origine. Si ha quindi a che fare con delle vere e proprie restrizioni a livello di "disposizione" che gli input devono assumere affinchè la dicotomia risulti apprendibile.

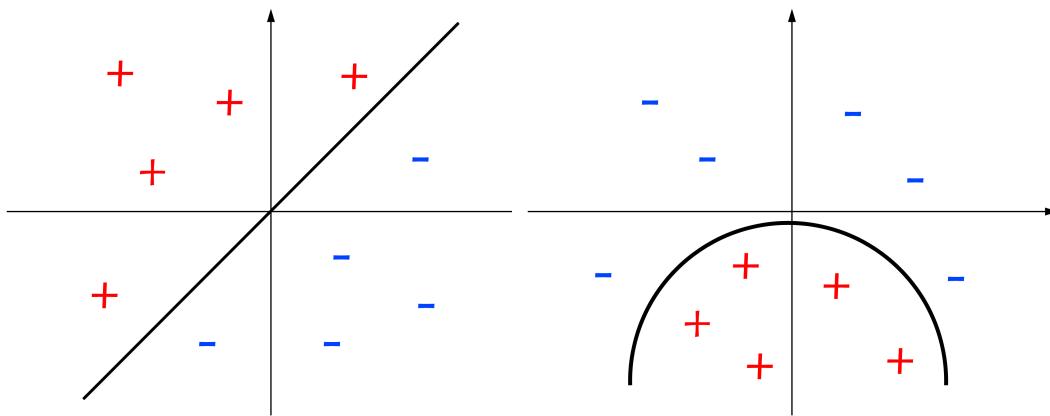


Figura 2.1: Confronto tra due set di input che ammettono una dicotomia linearmente separabile (a sinistra) e una non lineare (a destra). Un perceptron è in grado di apprendere solo la dicotomia di sinistra, mentre una rete più complessa potrebbe apprenderle entrambe

Per queste ragioni e per la "funzione" separatrice di dati, il perceptron è una rete neurale utilizzata come ultimo layer di reti più elaborate. Intuitivamente, si potrebbe pensare che il numero di dicotomie linearmente separabili dipenda dal particolare set di input che viene considerato. In realtà questo valore dipende soltanto dal numero di input p e dalle dimensioni dello spazio delle fasi n , introducendo però, l'ipotesi della *General Position*.

Tale condizione prevede che l'insieme di input $\{\xi^i, \dots, \xi^p\}$ non abbia alcun sottoinsieme di dimensione maggiore o uguale a n che sia linearmente dipendente. Esplicitandone il significato, la general position è un'estensione del concetto di indipendenza lineare: la richiesta infatti equivale al fatto che non esista alcun sottoinsieme di punti giacenti tutti sullo stesso iperpiano. Se, ad esempio, si raffigura nello spazio tridimensionale ($n = 3$) un insieme di $p > 3$ punti, di cui 3 di questi posti sullo stesso piano, si nota che la condizione della general position è fondamentale: la dicotomia sarebbe irrealizzabile a meno di attribuire lo stesso label a questi 3 punti.

Da quest'ultimo fatto sembrerebbe che il numero delle dicotomie linearmente separabili dipenda dalla disposizione dei punti dello spazio; la general position garantisce proprio il contrario e in aggiunta è una condizione facilmente verificabile, in quanto la probabilità che dei punti a caso giacciono sullo stesso iperpiano converge rapidamente a 0 all'aumentare delle dimensioni dello spazio.

2.1.2 Probabilità di apprendimento: dicotomie totali e linearmente separabili

L'obiettivo di una rete neurale sviluppata per il machine learning è quello di apprendere una "correlazione" tra input e label: nel caso di un perceptron l'apprendimento avviene nel caso in cui si abbia a che fare con una dicotomia linearmente separabile. È quindi importante capire quante siano le dicotomie di questo tipo in modo da comprendere quanto frequentemente questo tipo di rete neurale risulti efficace a risolvere il compito che le viene assegnato.

Ancora più nel particolare si è interessati allo studio della probabilità di apprendimento da parte di un perceptron durante la fase di addestramento: fissato un Training Set $\{\vec{\xi}^i, \sigma_i\}_{i=1,\dots,p}$ composto da p vettori in \mathbb{R}^n che soddisfano l'ipotesi della *General Position* e i relativi label σ_i , si studia la frazione del numero di dicotomie linearmente realizzabili rispetto al numero di dicotomie totali.

Ossia, scelta a caso una dicotomia tra quelle possibili, si studia con che probabilità essa sia linearmente separabile, cioè che sia realizzata e appresa da un perceptron. L'apprendimento di una dicotomia avviene quando gli output che la rete neurale assegna agli input $\{\xi^i, \dots, \xi^p\}$ coincidono con i label σ del training set.

Il numero delle dicotomie *possibili* per un insieme di p punti in n dimensioni corrisponde al numero di elementi dell'insieme delle parti.

Le nozioni di insiemistica affermano che se un insieme ha cardinalità p , allora il suo insieme delle parti avrà cardinalità 2^p . Da ciò ne consegue:

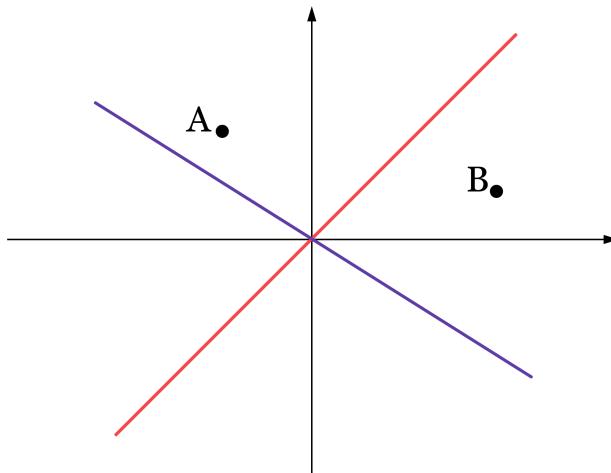
$$DicotomiePossibili = 2^p \quad (2.1.1)$$

Probabilisticamente, individuando il rapporto tra casi favorevoli e casi possibili, si definisce la probabilità di apprendimento:

$$\boxed{Prob = \frac{C(n, p)}{2^p}} \quad (2.1.2)$$

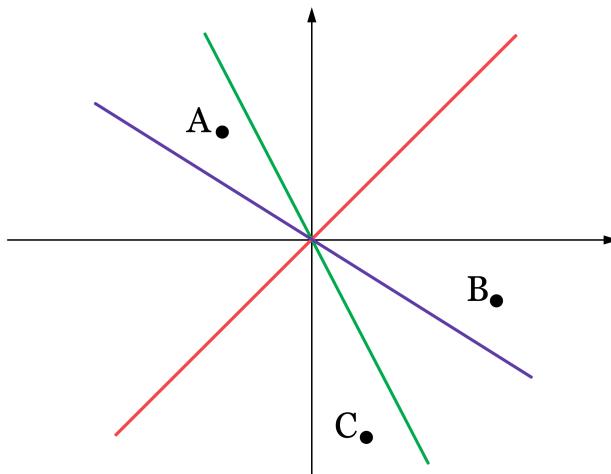
dove con $C(n, p)$ si indica il numero di dicotomie linearmente separabili realizzabili e in generale vale, $C(n, p) \leq 2^p$.

In generale il numero delle dicotomie linearmente separabili aumenta sì all'aumentare del numero di punti p , ma molto più lentamente di quanto accade per il numero di dicotomie totali, poiché solo una frazione di quelle possibili è linearmente realizzabile.



Dicotomie su $p = 2$ punti in $n = 2$ dimensioni (gli iperpiani sono rappresentati da delle rette): in questo caso sono realizzate 4 dicotomie totali, tutte linearmente separabili.

- { AB = ++ e AB = -- } identificate dall'iperpiano viola
- { AB = +- e AB = -+ } identificate dall'iperpiano rosso



Dicotomie su $p = 3$ punti in $n = 2$ dimensioni : in questo caso sono realizzate 8 dicotomie totali, ma *non* tutte linearmente separabili.

- { ABC = +-- e ABC = -++ } identificate dall'iperpiano rosso;
- { ABC = +-+ e ABC = -+- } identificate dall'iperpiano viola;
- { ABC = ++- e ABC = -+- } identificate dall'iperpiano verde.

Sono le 6 dicotomie realizzabili da un perceptron; le altre due, { ABC = +++ e ABC = --- } non sono realizzabili mediante un iperpiano passante per l'origine, si avrebbe bisogno ad esempio di una curva.

Per determinare il numero di dicotomie linearmente separabili, si fa riferimento allo studio di Cover [4] il cui risultato principale è il *Function Counting Theorem*.

2.2 Cover e il Function Counting Theorem

Dati p vettori in \mathbb{R}^n in *General Position*:

$$C(n, p) = 2 \sum_{k=0}^{n-1} \binom{p-1}{k} \quad (2.2.1)$$

$$\text{dove } \binom{\alpha}{\beta} = \begin{cases} \frac{\alpha!}{\beta!(\alpha-\beta)!} & \text{se } \alpha \geq \beta \\ 0 & \text{se } \alpha < \beta \vee \beta < 0 \end{cases}$$

$\binom{\alpha}{\beta}$ esprime il numero di sottoinsiemi di β elementi che si possono formare a partire da un insieme di α elementi. Dalla relazione del Binomial Theorem

$$\sum_{k=0}^n \binom{n}{k} = 2^n$$

si evince un altro metodo di rappresentazione del coefficiente binomiale: $\binom{n}{k}$ rappresenta infatti il coefficiente k -esimo alla riga p -esima del Triangolo di Tartaglia, la cui somma sulle righe è proprio 2^p .

Dimostrazione con il calcolo combinatorio In primo luogo si ricerca una formula di ricorrenza per il calcolo di $C(n, p)$.

Fissati $p - 1$ input in n dimensioni, si identifica con $C(n, p - 1)$, il numero dicotomie linearmente separabili realizzate da un Perceptron. Se si aggiunge un ulteriore punto ξ^p , ognuna delle $C(n, p - 1)$ dicotomie considerate assegnerà almeno un output al nuovo input, in modo correlato alla regione dello spazio in cui quest'ultimo è stato aggiunto. In generale quindi $C(n, p) \geq C(n, p - 1)$.

Fisso una delle $C(n, p - 1)$ dicotomie linearmente separabili e, di conseguenza, fissò tutti i possibili iperpiani che suddividono lo spazio delle fasi.

Quando il p -esimo input è collocato avvengono principalmente due situazioni:

- ξ^p si trova in una regione dello spazio in cui non passa alcuno dei possibili iperpiani correlati alla dicotomia fissata (zone bianche della figura (2.2)).

In questo caso gli iperpiani che separano gli input rimarranno gli stessi, di conseguenza l'output di questo p -esimo punto sarà vincolato, in quanto l'unica dicotomia linearmente separabile possibile sarà quella fissata inizialmente.

- ξ^p è attraversato da uno dei possibili iperpiani correlati alla dicotomia fissata (zone gialle della figura (2.2)).

In questa situazione gli iperpiani fissati non realizzano più la dicotomia iniziale: l'insieme di questi ultimi passanti al di "sopra" del punto ξ^p realizzerà un'ulteriore dicotomia e lo stesso accadrà per la famiglia di iperpiani passanti al di "sotto" di esso.

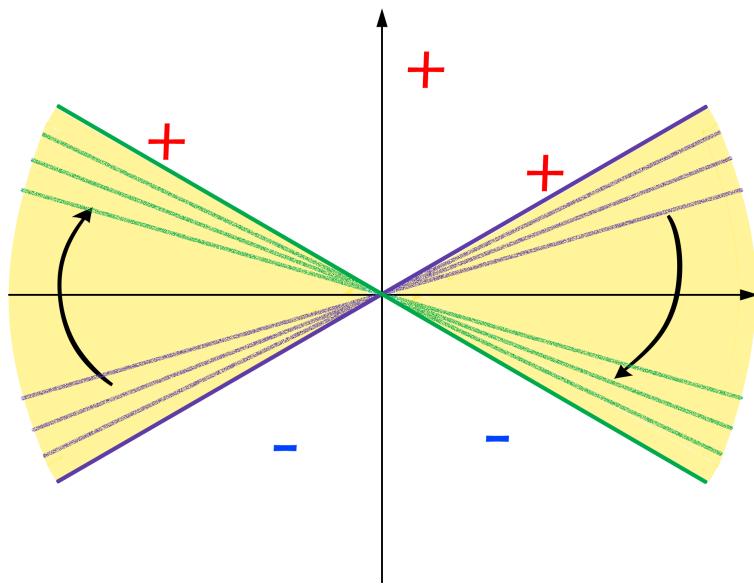


Figura 2.2

Si è quindi in grado di esprimere la variazione del numero di dicotomie $C(n, p-1)$ all'aggiunta del p -esimo punto; chiamando infatti D il numero di dicotomie che ricadono nel caso 2, si ha:

$$C(n-p) - C(n, p-1) = D \quad (2.2.2)$$

D rappresenta quindi il numero di dicotomie in n dimensioni su $p-1$ punti, identificate da almeno un iper piano passante per il p -esimo punto ξ^p .

Imporre questo passaggio equivale però a vincolare la dimensione n-esima dello spazio ambiente, arrivando cioè a definire:

$$D = C(n - 1, p - 1) \quad (2.2.3)$$

Inserendo l'equazione appena ottenuta in (2.2.2) si ottiene, infine, la formula di ricorrenza per il calcolo di $C(n, p)$:

$$\boxed{C(n, p) = C(n, p - 1) + C(n - 1, p - 1)} \quad (2.2.4)$$

Supponendo la validità della tesi del teorema (2.2.1), mostro che essa rispetta la relazione di ricorrenza appena trovata:

$$C(n, p) = C(n, p - 1) + C(n - 1, p - 1) \quad (2.2.5)$$

$$= 2 \sum_{k=0}^{n-1} \binom{p-2}{k} + 2 \sum_{k=0}^{n-2} \binom{p-2}{k} \quad (2.2.6)$$

$$= 2 \sum_{k=0}^{n-1} \binom{p-2}{k} + 2 \sum_{k=0}^{n-1} \binom{p-2}{k-1} \quad (2.2.7)$$

$$= 2 \sum_{k=0}^{n-1} [\binom{p-2}{k} + \binom{p-2}{k-1}] = 2 \sum_{k=0}^{n-1} \binom{p-1}{k} \quad (2.2.8)$$

■

dove in (2.2.8) si ha usato la proprietà: $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$
e in (2.2.7) la convenzione che $\binom{n}{k} = 0$ per $k < 0$

Dimostrazione e conseguenze

Si consideri l'equazione di ricorrenza ottenuta (2.2.4), essa ha due condizioni al contorno:

1.

$$C(0, p) = 0 \quad \forall p \quad (2.2.9)$$

Caso "degenero" di un perceptron con 0 neuroni;

2.

$$C(n, 1) = 2 \quad \forall n \neq 0 \quad (2.2.10)$$

Un solo punto nello spazio delle fasi dà "origine" a due dicotomie linearmente separabili.

Cercando una soluzione all'equazione di ricorrenza (2.2.4), si rappresenta la situazione in un "reticolo" formato da due assi principali, l'asse orizzontale n con valori interi da 0 in poi (numero di neuroni $n \geq 0$) e l'asse verticale p con valori interi da 1 in poi (numero vettori di input $p \geq 0$).

Ogni valore di $C(n, p)$ sarà rappresentato da un punto del reticolo e avrà come coordinate in "ascissa" il valore di n e in "ordinata" il valore di p .

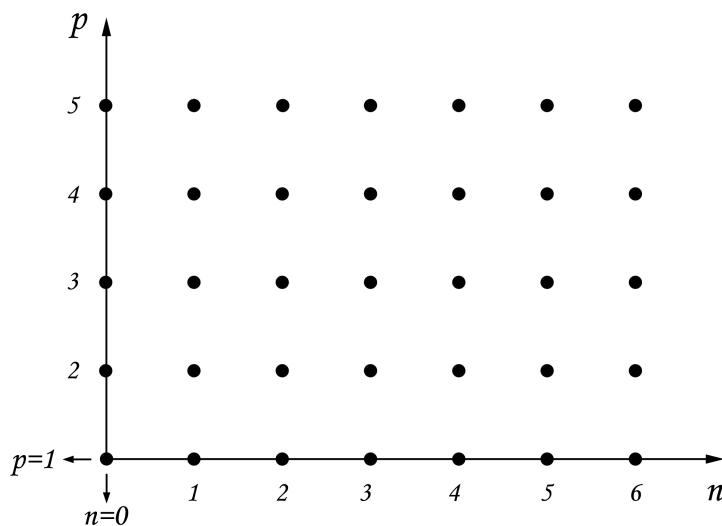


Figura 2.3

Subito si osserva che le condizioni al contorno (2.2.9) e (2.2.10), sono rappresentate dai punti dell'asse orizzontale n , e immaginando di voler determinare uno specifico valore di $C(n, p)$ se ne studia il propagarsi dei loro contributi.

Ad esempio, si vuole determinare il valore di $C(6, 5)$ identificato in figura con punto del reticolo cerchiato in rosso, dall'equazione di ricorrenza (2.2.4) si ha quindi che $C(6, 4)$ e $C(5, 4)$ (punti cerchiati in blu) sono due contributi. Ognuno di essi a sua volta dipenderà da alcuni $C(n, p)$ della riga "sottostante" (punti cerchiati in verde), in particolare quelli dove il collegamento è rappresentato da un "movimento":

1. Verso il basso sulla stessa colonna verticale, termine $C(n, p - 1)$ dell'equazione (2.2.4);
2. Verso il diagonalmente in basso a destra, termine $C(n - 1, p - 1)$ dell'equazione (2.2.4).

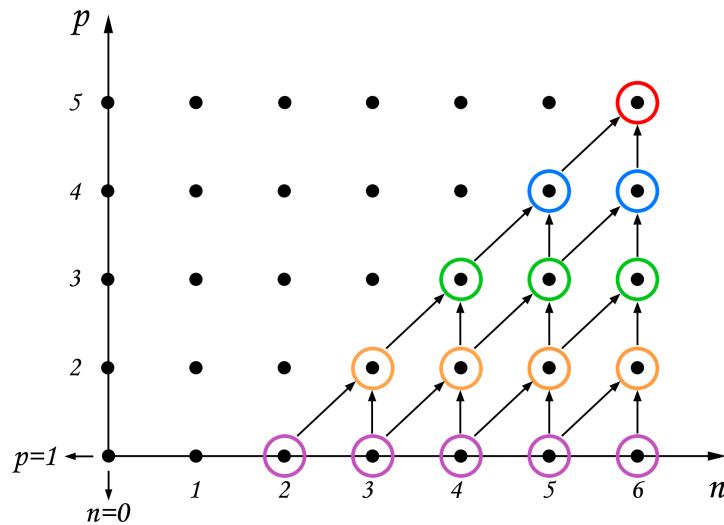


Figura 2.4: Contributi al calcolo di $C(6,5)$: dai termini di bordo rappresentati in viola, le frecce rappresentano tutti i possibili movimenti

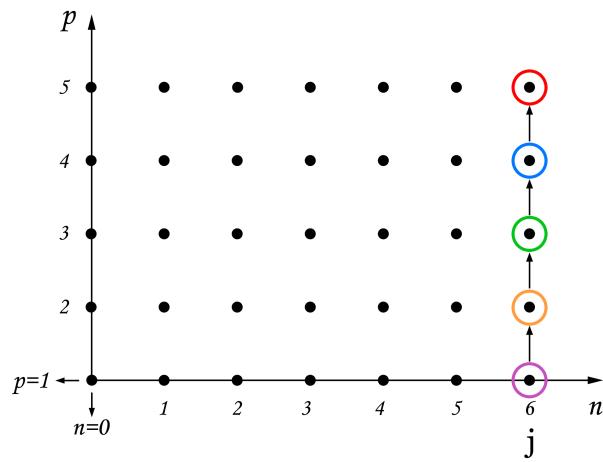
Si procede in questo modo fino ad ottenere i termini corrispondenti alle condizioni al contorno, ovvero fino ad arrivare alla "riga" corrispondente a $p = 1$, ovvero: $C(j, 1)$ di cui se ne conosce il valore preciso grazie a (2.2.9) e (2.2.10).

In generale, con questo procedimento, si ha che le condizioni al contorno ad altezza $p = 1$ e il valore di $C(n, p)$ da determinare (il quale, appunto, si trova ad altezza p), sono collegati attraverso *cammini* di $p - 1$ passi con le seguenti proprietà:

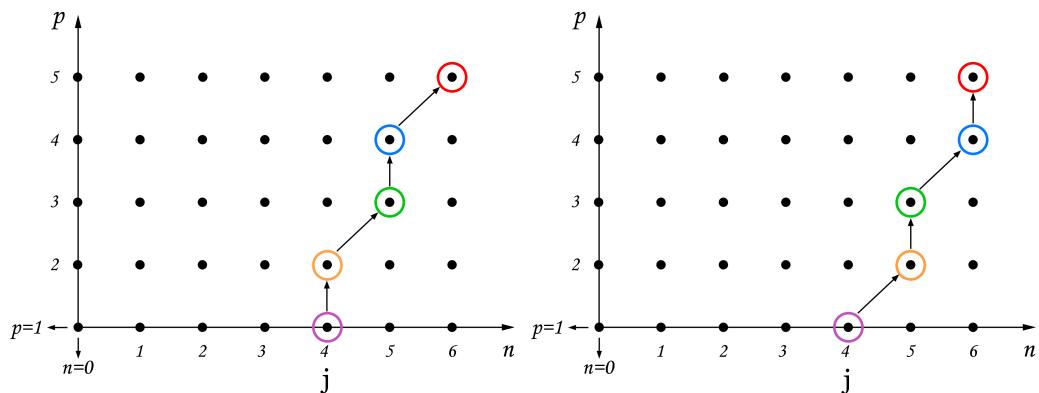
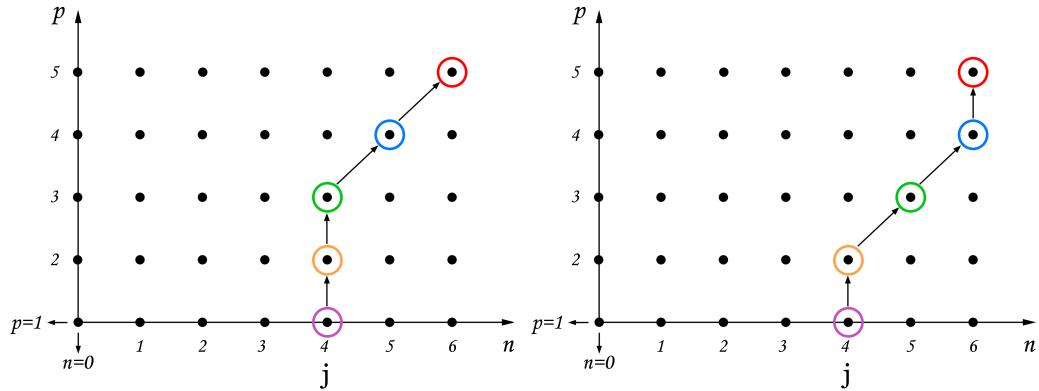
- Ogni passo è verso la riga superiore $p + 1$
- Il passo può essere rappresentato da un movimento verticale sulla stessa colonna (stesso indice n);
- Il passo può essere rappresentato da un movimento diagonale verso l'altro a destra, (indice $n + 1$).

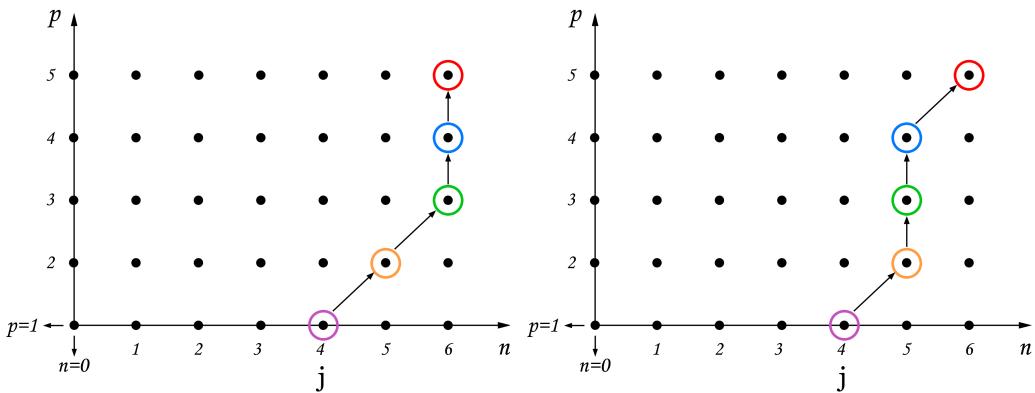
Per determinare tutte le condizioni al contorno $C(j, 1)$ che contribuiscono ad uno specifico $C(n, p)$ cerco il numero di *cammini diretti* di $p - 1$ passi e con le proprietà sopra elencate, per andare da uno all'altro.

Se $j = n$ ho un unico cammino possibile:



Se invece, $j \neq n$, ad esempio $j = 4$, i possibili cammini sono diversi: ad esempio, come nelle figure, per andare da $C(4,1)$ a $C(6,5)$ con $p - 1 = 4$ passi, ci sono 6 modi differenti, 6 cammini diretti.





Dei 4 passi che compongono il cammino, 2 sono vincolati ad essere diagonalmente verso destra, in quanto nel complesso n deve aumentare di 2, partendo da $C(4, 1)$ e dovendo arrivare a $C(6, 5)$. I due steps rimanenti, di conseguenza, devono essere rappresentati da un movimento verticale dove n rimane costante.

Qualsiasi sia l'ordine in cui si susseguono i 4 passi, l'unico vincolo è che 2 siano diagonali in alto a destra: per definizione nel calcolo combinatorio, questo fatto si può rappresentare con il *coefficiente binomiale* e, proprio come ottenuto dalla rappresentazione del reticolo in figura, si ha che i cammini diretti sono:

$$\binom{4}{2} = \frac{4!}{2! \cdot (4-2)!} = 6 \quad (2.2.11)$$

Estendendo al caso generale, si ha che per andare da $C(j, 1)$ a $C(n, p)$:

- Il cammino diretto è composto da $p - 1$ passi;
- Dei $p - 1$ passi, $n - j$ sono vincolati ("movimenti" diagonali in alto a destra)

Ovvero in numero i cammini saranno pari a $\binom{p-1}{n-j}$ ed esprimono quante volte una data condizione al contorno, il cui valore si propaga ricorsivamente, contribuisce al valore di $C(n, p)$. Tutte le condizioni al contorno $C(j, 1)$ però, propagano un valore diverso, infatti si ha che:

- I termini con $j = 0$, $C(0, 1) = 0$ non contribuiscono al valore di $C(n, p)$ (come si evince in (2.2.9));
- Tutte le altre condizioni contribuiscono con valore 2: $C(j, 1) = 2 \quad \forall j \neq 0$

Moltiplicando il valore del contributo della j -esima condizione al contorno per il numero di cammini diretti di $p - 1$ passi che la collegano a $C(n, p)$, si ottiene quindi la relazione:

$$C(n, p) = \sum_{j=1}^n 2 \cdot \binom{p-1}{n-j} \quad (2.2.12)$$

(la sommatoria inferiore ha indice $j = 1$ in quanto si è osservato che è nullo il contributo propagato dalla condizione per $j=0$).

Inoltre ricordando che il coefficiente binomiale ha come proprietà $\binom{\alpha}{\beta} = 0$ se $\alpha < \beta$, alcuni termini della sommatoria saranno nulli: essa si arresterà quindi a $n - k = p - 1$.

Infine, effettuando il cambio di indice $n - j = k$, si ha che:

$$C(n, p) = 2 \cdot \sum_{j=1}^n \binom{p-1}{n-j} = 2 \cdot \sum_{k=0}^{n-1} \binom{p-1}{k} \quad (2.2.13)$$

Si è quindi ottenuta la relazione del Function Counting Theorem (2.2.1).

Conseguenze del teorema La formula (2.2.1) ottenuta da Cover per il calcolo del numero di dicotomie linearmente realizzabili ha interessanti conseguenze.

Apprendimento con eccedenza di neuroni

Nel caso in cui $p \leq n$ la somma sull'indice k è superiormente limitata e si arresta a $p - 1$, ottenendo quindi:

$$C(n, p) = 2 \cdot \sum_{k=0}^{p-1} \binom{p-1}{k} = 2 \cdot 2^{p-1} = 2^p \quad (2.2.14)$$

ovvero il numero di dicotomie linearmente realizzabili coincide con il numero totale di dicotomie e la probabilità di apprendimento sarà pari a 1.

Questa conclusione è abbastanza intuitiva se si pensa che, avendo meno di un punto per ogni dimensione, è sempre possibile trovare un iperpiano che realizzi ogni dicotomia.

Scarso apprendimento con pochi neuroni

Un'altra importante conseguenza si ottiene dall'analisi del caso $p \gg n$, ovvero se il numero di input che si vuole classificare è molto maggiore del numero di neuroni.

Per studiare il comportamento della sommatoria, si utilizza una proprietà del triangolo di Tartaglia, ricordando che esso è un espediente grafico per visualizzare il

coefficiente binomiale. Gli elementi $\binom{p}{k}$ sono monotoni crescenti in k fino a $k \leq \frac{p}{2}$, cioè fino a metà della riga del triangolo di Tartaglia, e simmetricamente decrescenti per $k \geq \frac{p}{2}$. La condizione $p \gg n$ implica che ci si trova a sinistra dell'asse di simmetria del triangolo, pertanto, la sommatoria può essere maggiorata e minorata

$$2 \cdot \binom{p-1}{n-1} < 2 \cdot \sum_{k=0}^{n-1} \binom{p-1}{k} < 2 \cdot (n-1) \binom{p-1}{n-1} \quad (2.2.15)$$

in modo che al primo membro si conti soltanto il contributo dell'ultimo indice della somma e all'ultimo membro si conti $(n-1)$ volte il contributo dell'ultimo indice della somma.

Studiando il coefficiente binomiale tramite la sua esplicitazione con i fattoriali si ha che:

$$\binom{p-1}{n-1} = \frac{(p-1)!}{(n-1)!(p-n)!} = \frac{1}{(n-1)!} (p-1)(p-2) \dots (p-n+1) = \frac{1}{(n-1)!} (p^{n-1} + \mathcal{O}(p^{n-2})) \quad (2.2.16)$$

Questa equazione mostra che sia la maggiorazione sia la minorazione della sommatoria hanno lo stesso andamento, a meno di una costante moltiplicativa dipendente da n . Per questa ragione, ad n fissato, per $p \gg n$ si può assumere:

$$C(n, p) \sim A \cdot p^{n-1} \quad (2.2.17)$$

Osservando ora la probabilità di apprendimento

$$Prob = \frac{C(n, p)}{2^p} \sim \frac{p^{n-1}}{2^p} \xrightarrow{p \rightarrow \infty} 0 \quad (2.2.18)$$

si evince che al crescere del numero di input, la probabilità che una certa dicotomia venga appresa da una rete con n neuroni converge a 0.

Ad eccezione del limite finale, i calcoli effettuati valgono purchè la sommatoria in (2.2.1) si arresti prima di raggiungere l'asse di simmetria del triangolo di Tartaglia, così da legittimare la minorazione e la maggiorazione effettuate. Tale condizione è analiticamente espressa con $n \leq \frac{p}{2}$.

Ci si può quindi chiedere a questo punto cosa accada se $\frac{p}{2} < n < p$.

Il triangolo di Tartaglia è simmetrico rispetto alla verticale (cioè vale $\binom{p}{k} = \binom{p}{p-k}$) e la sommatoria in (2.2.1) rappresenta la somma della riga $(p-1)$ -esima dei primi $(n-1)$ coefficienti. Tale somma è la somma dell'intera riga (che si sa essere 2^p) meno la somma dei termini dal n -esimo in poi. Per differenza, e per ciò che si è

ottenuto precedentemente, si può valutare che per $\frac{p}{2} < n < p$ si ha

$$C(n, p) \sim 2^p - A \cdot p^{n-1} \quad (2.2.19)$$

valutandone il limite per $p \rightarrow +\infty$ (in questo caso anche $n \rightarrow +\infty$ per soddisfare l'ipotesi) si ottiene:

$$Prob \sim \frac{2^p - p^{n-1}}{2^p} \xrightarrow{p \rightarrow \infty} 1 \quad (2.2.20)$$

Configurazione intermedia

Le conseguenze che si sono analizzate precedentemente hanno comportamenti opposti. La configurazione *critica* che determina il rapido cambio di comportamento si ha per $p = 2n$, ovvero quando i vettori di input p sono il doppio rispetto al numero di neuroni n . In questa circostanza si può calcolare facilmente $C(n, p)$, in quanto la sommatoria di (2.2.1) si arresta a metà riga del triangolo di Tartaglia. Sfruttando la proprietà di simmetria di quest'ultimo, secondo la quale la somma su metà riga deve coincidere con la metà della somma della riga, si ha:

$$C(n, p) = C(n, 2n) = 2 \cdot \sum_{k=0}^{n-1} \binom{2n-1}{k} = 2^{p-1} \quad (2.2.21)$$

ovvero che il numero delle dicotomie linearmente realizzabili sarà pari alla metà di tutte le dicotomie possibili, indipendentemente da n . Conseguentemente, valutando la probabilità di *learning* in queste condizioni,

$$Prob = \frac{1}{2} \quad (2.2.22)$$

si ottiene che, scelta a caso una dicotomia tra le possibili, il perceptron avrà il 50% di apprenderla, a patto che abbia a disposizione un numero di neuroni pari alla metà dei dati da classificare.

2.2.1 Un parametro chiave: la capacità

Studiando la probabilità di apprendimento da parte di una rete neurale si definisce una grandezza rappresentativa della misura della quantità di informazioni che una rete neurale può immagazzinare. È detto *load* il rapporto tra il numero di input e il numero di neuroni della rete.

$$load = \frac{p}{n} \quad (2.2.23)$$

2.2. COVER E IL FUNCTION COUNTING THEOREM

esso rappresenta una misura della densità di informazioni per grado di libertà che si prova a far apprendere alla rete neurale.

Inoltre valutandone l'andamento si arriva ad un parametro chiave, la cosiddetta *storage capacity* o capacità α_c , valore critico del load oltre il quale scelta a caso una dicotomia tra le possibili, la probabilità valutata nel limite termodinamico che essa sia realizzata e appresa dalla rete tende a zero.

α_c si dice misurare l'*espressività* della rete neurale e corrisponde al valore in cui l'apprendimento di una dicotomia è del 50%. Più alto è questo valore maggiore sarà la capacità espressiva della rete e, conseguentemente, il numero di dati che la rete neurale è in grado di classificare sarà maggiore. Studio la capacità per il perceptron: ovvero cerco il valore di $n^* = \frac{p}{\alpha_c}$ tale che

$$Prob = \frac{C(\frac{p}{\alpha_c}, p)}{2^p} = \frac{1}{2} \quad (2.2.24)$$

Questa richiesta è rispettata, come mostrato in 2.2.21, quando $p = 2n$ e di conseguenza da questo si evince che:

$$\boxed{\alpha_c = 2} \quad (2.2.25)$$

Inoltre osservando le probabilità di apprendimento nei casi (2.2.14) e (2.2.18), ovvero rispettivamente i regimi con valore del load precedente e successivo al valore critico, ci si accorge come la capacità α_c sia la soglia oltre la quale il perceptron non è più in grado di "imparare", proprio come si evince dal grafico successivo:

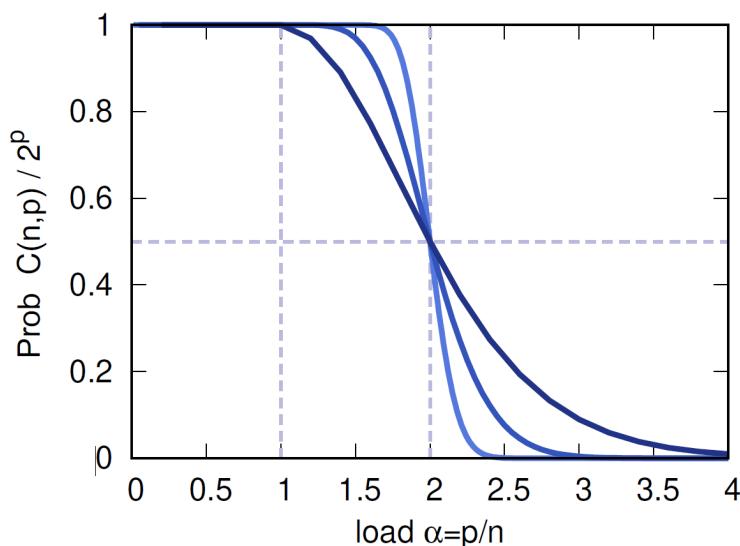


Figura 2.5

Tutte queste considerazioni possono essere osservate tramite la rappresentazione in "reticolo" usate nella sezione 2.2. Come in figura, osservando tutti i contributi da parte delle condizioni al contorno ad uno specifico valore di $C(n, p)$, si determina un "cono di influenza" (evidenziato in giallo) costituito da tutti i punti attraversati da un cammino diretto che ha come punto iniziale una delle p condizioni al contorno alla base del cono e come punto finale $C(n, p)$.

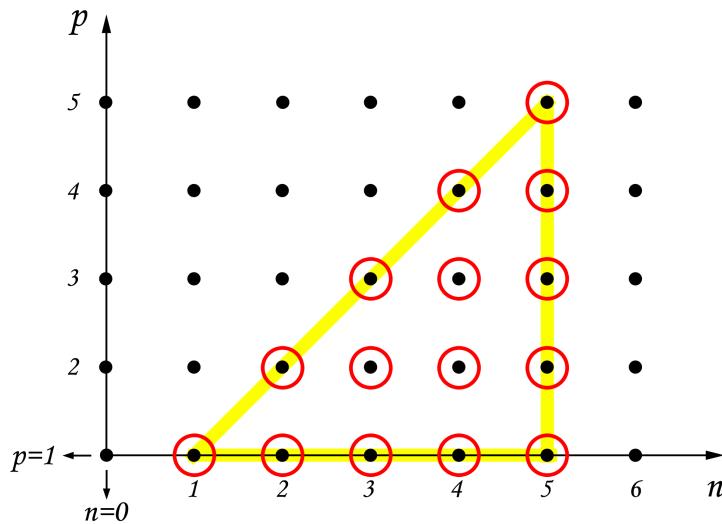


Figura 2.6

Anche in questo caso i cammini sono composti solamente da tratti verticali sulla stessa colonna o diagonali verso l'alto a destra. Essendo le condizioni al contorno per input puntiforme (2.2.9) e (2.2.10), ed essendo la base del cono costituita da p termini, si ha che $C(n, p)$ ha come valore massimo 2^p .

Ciò però accade solo quando *tutta* la base del cono di influenza, tutte le condizioni al contorno alla riga $p = 1$, $C(n, 1)$ hanno $n > 0$, corrispondendo cioè a condizioni al contorno con contributo pari a 2. Dalla rappresentazione in reticolo osservo che questa situazione accade per $p \leq n$ (come in figura (2.6)), in questo caso, proprio come dedotto in (2.2.14):

$$Prob = \frac{C(n, p)}{2^p} = \frac{2^p}{2^p} = 1 \quad (2.2.26)$$

ovvero tutte le dicotomie saranno linearmente separabili, realizzate e apprese da un perceptron. Se al contrario $p > n$, non tutta la base del cono di influenza ha $n > 0$, al valore di $C(n, p)$ contribuiranno anche condizioni al contorno con valore nullo.

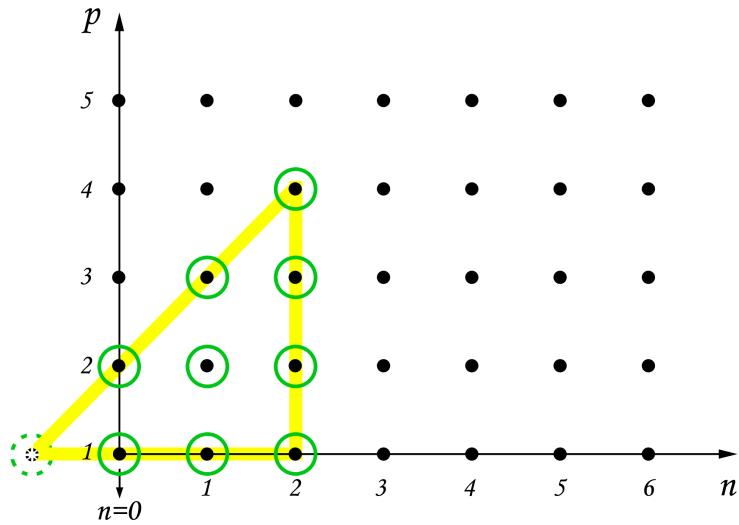


Figura 2.7: Calcolo di $C(2, 4)$: in questo specifico caso, alla base del cono sono presenti anche $C(0, 1)$ e "C($-1, 1$)", le quali hanno per definizione contributo nullo.

Di conseguenza se $p > n \Rightarrow C(n, p) < 2^p$ e $\text{Prob} = \frac{C(n, p)}{2^p} < 1$.

In questo regime, non tutte le dicotomie quindi, saranno apprese da un perceptron.

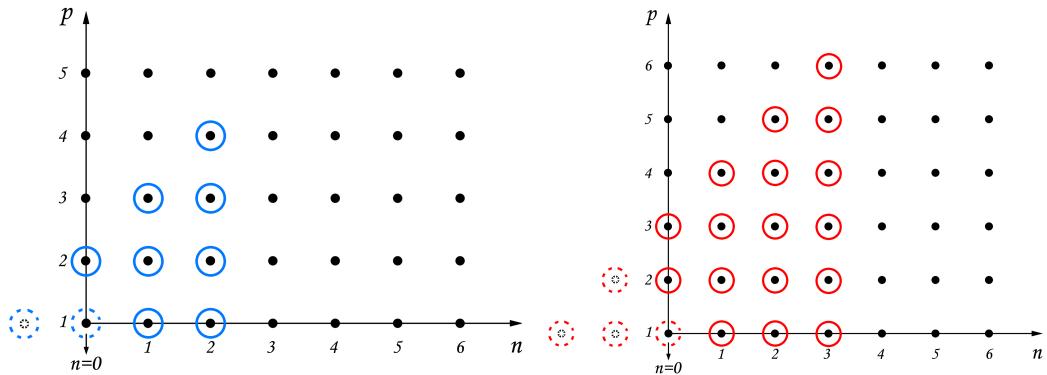


Figura 2.8: A sinistra: calcolo di $C(2, 4)$: 4 condizioni al contorno, di cui 2 con contributo nullo (cerchi tratteggiati) e 2 con contributo non nullo.

A destra: calcolo di $C(3, 6)$: 6 condizioni al contorno, di cui 3 con contributo nullo (cerchi tratteggiati) e 3 con contributo non nullo

Indagando per arrivare alla capacità, ovvero il valore critico $\alpha_c = \frac{p}{n}$ corrispondente ad una probabilità di apprendimento $\frac{1}{2}$, si nota che ciò si verifica

se $C(n, p) = \frac{2^p}{2}$, ossia solo quando *la metà* delle condizioni al contorno, la metà della base del cono di influenza avrà termini con contributo pari a 2, avendo $n > 0$. Questo coincide al fatto che solo $\frac{p}{2}$ delle condizioni al contorno avrà $n > 0$, proprio come in figura (2.8).

Essendo $C(n, 1)$ il "limite destro" della base del cono di influenza, la condizione al contorno che ha per unico cammino diretto quello costituito solo da movimenti verticali, il requisito di avere metà base con $n > 0$ corrisponde a:

$$\frac{p}{2} = n \Rightarrow \boxed{p = 2n} \quad (2.2.27)$$

$$\text{Per } p = 2n, C(n, p) = C(n, 2n) = \frac{2^p}{2} \Rightarrow \text{Prob} = \frac{2^{p-1}}{2^p} = \frac{1}{2}.$$

Si è quindi ottenuto il valore critico del load $\frac{p}{n}$ per input puntiformi, proprio come nella relazione (2.2.25):

$$\boxed{\alpha_c = 2} \quad (2.2.28)$$

2.3 Probabilità di separazione per doppietti monodispersi

La teoria di Cover sulle dicotomie di input puntiformi è stato un risultato fondamentale per comprendere meglio ciò che riguarda la meccanica statistica delle reti neurali. Si è infatti ottenuta una formula esatta per determinare $C(n, p)$ a valori finiti, anche piccoli di n e di p .

Seguendo quest'ottica il lavoro [1] ha lo scopo di estendere questa teoria analitica anche al caso in cui gli input non siano puntiformi, ovvero politopi a distanza fissata o con specifiche strutture geometriche.

Quando si considerano situazioni reali infatti, alcune delle ipotesi riguardanti i dati in input costituenti il training set potrebbero essere troppo restrittive o comunque identificare casi fin troppo "ideali". Questo elaborato di tesi si occuperà di trattare il caso in cui il training set sia composto da coppie di punti a distanza fissata, p politopi, equivalentemente chiamati *doppietti*. L'insieme sarà identificato con $\{\xi^i, \bar{\xi}^i\}_{i=1,\dots,p}$ e la distanza presa in considerazione non è quella euclidea, bensì la distanza angolare tra i punti del politopo, anche chiamata *overlap* e definita come:

$$\rho = \frac{\xi \cdot \bar{\xi}}{\|\xi\| \cdot \|\bar{\xi}\|} = \cos(\vartheta) \quad (2.3.1)$$

dove ϑ è l'angolo compreso tra i punti del doppietto.

La distanza così definita avrà la proprietà $\rho \in [-1, 1]$: subito si evince che se $\rho = 1 \rightarrow \cos(\vartheta) = 1 \rightarrow \vartheta = 0$ ovvero il doppietto si riconduce alla situazione di input puntiformi. Come in precedenza, il fatto che le dicotomie siano lineari permette sia di avere ξ e $\bar{\xi}$ normalizzati sull'ipersfera di raggio unitario, sia di ragionare in termini di distanza angolare tra essi, senza perdite di generalità.

2.3.1 Solo una parte delle dicotomie è realizzabile: la grandezza Ψ

Il fatto che gli input siano organizzati a doppietti comporta un diverso calcolo delle dicotomie realizzabili: si passa infatti da input con reciproca posizione indipendente, a coppie di dati, dove fissato ξ^i , il correlato $\bar{\xi}^i$ sarà vincolato al luogo dei punti a distanza ρ da ξ^i . Inoltre elementi dello stesso doppietto saranno classificati allo stesso modo, a differenza di input puntiformi, la cui reciproca classificazione è totalmente

2.3. PROBABILITÀ DI SEPARAZIONE PER DOPPIETTI MONODISPERSI

indipendente. Si nota innanzitutto che un doppietto con $\rho = -1$ è costituito da punti dialmetralmente opposti, perciò le dicotomie linearmente realizzabili su di esso sono 0, perché i due punti non potranno mai essere classificati allo stesso modo. Da qui in avanti si considereranno i cosiddetti valori *non critici*, ovvero dove l'overlap $\rho \in (-1, 1]$. Si pensi ad una situazione di due input in due dimensioni: si collocano ξ^1 e ξ^2 casualmente nel piano e successivamente $\bar{\xi}^1$ e $\bar{\xi}^2$ secondo i vincoli dati da ρ (fissato precedentemente).

Può accadere che i doppietti si "sovrappongano", pertanto il numero di dicotomie linearmente realizzabili sarà minore del valore che si aveva per input puntiformi e, come altra conseguenza si avrà una qualche dipendenza da ρ (doppietti più "grandi" saranno più facilmente sovrapponibili). Per questa ragione si arriva all'introduzione della quantità Ψ , che quantifica la frazione, il numero di iperpiani nello spazio, che assegnano lo stesso *label* ad entrambi i punti del doppietto. Nell'articolo [1] e in appendice (6.1) Ψ_2 viene calcolata tramite un integrale che tiene conto del numero di iperpiani che realizzano la condizione richiesta, rispetto al numero totale di iperpiani nello spazio. Dai calcoli emerge che questa grandezza è funzione soltanto dell'overlap ed è esprimibile come:

$$\Psi_2(\rho) = \frac{2}{\pi} \cdot \arctan \sqrt{\frac{1+\rho}{1-\rho}} \quad (2.3.2)$$

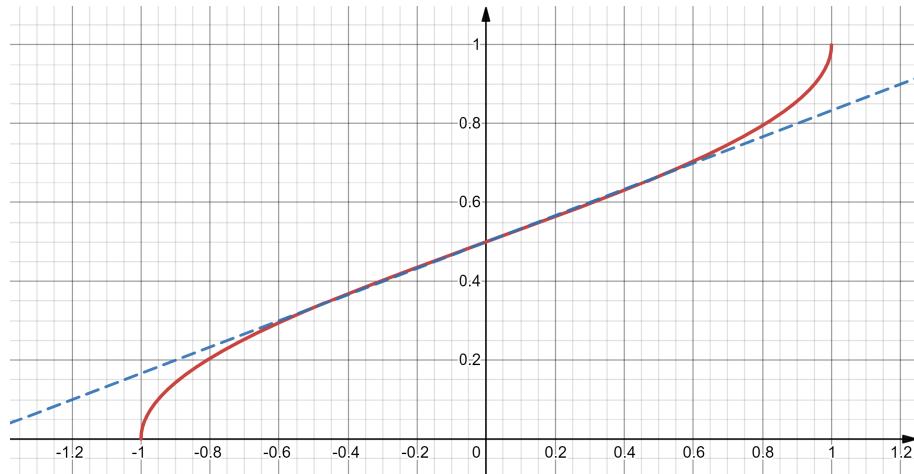


Figura 2.9: In rosso grafico della funzione $\Psi_2(\rho)$. Nell'intervallo $\rho \in [-0.7, 0.7]$ tale funzione è approssimabile alla retta tratteggiata in blu, di equazione $y = \frac{x}{3} + \frac{1}{2}$

2.3.2 Dicotomie linearmente realizzabili su doppietti monodispersi

Si procede al calcolo del numero di dicotomie linearmente separabili realizzabili su un training set costituito da doppietti a distanza angolare fissata.

Il numero di dicotomie totali resta invariato rispetto al caso puntiforme, esso dipende solo dal numero di input, non dalla struttura geometrica né dalla disposizione che essi hanno nello spazio delle fasi.

$$DicotomiePossibili = 2^p \quad (2.3.3)$$

$C(n, p)$ per doppietti a distanza fissata, dipende da ρ attraverso $\Psi_2(\rho)$, il quale, nel nostro caso di studio è uguale per tutti i p doppietti. Si effettuerà un ragionamento analogo a quello che si è fatto per dimostrare il *Function Counting Theorem* avendo come obiettivo quello di trovare una formula di ricorrenza per il calcolo di $C(n, p, \rho)$.

In partenza si ha un insieme $\left\{ \{\xi^1, \bar{\xi}^1\}, \dots, \{\xi^{p-1}, \bar{\xi}^{p-1}\} \right\}$ di $p - 1$ doppietti in n dimensioni con overlap fissato ρ . Su di esso, con $C(n, p - 1, \rho)$ si identifica il numero di dicotomie linearmente separabili realizzate. Si aggiunge solamente il punto ξ^p e si applica il Function Counting Theorem.

Fisso una dicotomia delle $C(n, p - 1, \rho)$, se uno degli iperpiani che la identifica passa per l'ultimo punto aggiunto ξ^p , allora si "origineranno" due ulteriori dicotomie (una per ognuno dei possibili *label* che si assegna a ξ^p). Chiamato S il numero delle dicotomie tra le $C(n, p - 1, \rho)$ di questo tipo, si ha che $C(n, p - 1, \rho) - S$ sarà il numero di dicotomie non identificate da un piano passante per ξ^p .

Ne consegue che sul "nuovo" insieme $\left\{ \{\xi^1, \bar{\xi}^1\}, \dots, \{\xi^{p-1}, \bar{\xi}^{p-1}\}, \{\xi^p\} \right\}$, dette Q il numero di dicotomie linearmente separabili, S potrà essere inteso come la "variazione" del numero di dicotomie linearmente separabili in seguito all'aggiunta del punto ξ^p , ovvero:

$$Q - C(n, p - 1, \rho) = S \quad (2.3.4)$$

Da quest'ultima relazione, considerando la definizione di S , e osservando che imporre il passaggio per ξ^p equivale a vincolare la dimensione n -esima dello spazio ambiente (cioè che $S = C(n - 1, p - 1, \rho)$) , si ottiene:

$$Q = C(n, p - 1, \rho) - C(n - 1, p - 1, \rho) \quad (2.3.5)$$

Ora si aggiunge il *partner* $\bar{\xi}^p$. Iterando il ragionamento precedente chiamo R il numero di dicotomie tra le Q identificate da un iperpiano passante per $\bar{\xi}^p$; in questo

2.3. PROBABILITÀ DI SEPARAZIONE PER DOPPIETTI MONODISPERSI

caso però non si origineranno due ulteriori dicotomie, bensì solo una, in quanto il label di $\bar{\xi}^p$ sarà vincolato ad essere lo stesso del correlato ξ^p (per la proprietà per cui punti dello stesso doppietto saranno classificati allo stesso modo).

Delle $Q - R$ dicotomie (cioè quelle non identificate attraverso un iperpiano passante $\bar{\xi}^p$) *solo alcune* assegneranno il label corretto a $\bar{\xi}^p$: quelle dove ciò non accade, quindi, saranno da scartare. Mediamente la frazione di dicotomie che rispetta questa condizione è rappresentata da Ψ_2 , si giunge pertanto alla relazione:

$$C(n, p, \rho) = R + \Psi_2(\rho)(Q - R) \quad (2.3.6)$$

Tra le Q dicotomie (linearmente separabili su $p - 1$ doppietti e ξ^p), $S = C(n - 1, p - 1, \rho)$ sono identificate da un iperpiano passante per ξ^p e $Q - S = C(n, p - 1, \rho)$ sono svincolate da qualsiasi passaggio. Osservando una delle R dicotomie si può "ricadere" in due casi:

1. la dicotomia è identificata da un iperpiano passante solamente per $\bar{\xi}^p$ (in questo caso si vincola una sola dimensione dello spazio ambiente)
2. la dicotomia è identificata da un iperpiano passante sia per $\bar{\xi}^p$ sia per ξ^p (in questo caso si vincolano due dimensioni dello spazio ambiente)

Da questo ragionamento si ricava:

$$R = C(n - 1, p - 1, \rho) + C(n - 2, p, \rho) \quad (2.3.7)$$

(Quest'ultima relazione è ricavabile anche applicando il Function Counting Theorem su $p - 1$ doppietti con l'aggiunta del punto ξ^p , in $n - 1$ dimensioni dato il vincolo del passaggio per $\bar{\xi}^p$).

Unendo le relazioni (2.3.5) , (2.3.6) , (2.3.7) si arriva infine alla formula di ricorrenza per input costituiti da una struttura di doppietti a distanza fissata ρ :

$C(n, p, \rho) = \Psi_2 \cdot C(n, p - 1, \rho) + C(n - 1, p - 1, \rho) + (1 - \Psi_2) \cdot C(n - 2, p - 1, \rho)$

(2.3.8)

2.3.3 Come cambia la capacità per dati strutturati

Ottenuta la formula di ricorrenza per dati strutturati a doppietti con distanza fissata (2.3.8), si ha un'evidente difficoltà nel risolverla in forma chiusa.

Cercando un modo per estrapolare il valore e l'andamento della capacità α_c , proprio come nel caso di input puntiformi si effettua un ragionamento analogo a

2.3. PROBABILITÀ DI SEPARAZIONE PER DOPPIETTI MONODISPERSI

quello sviluppato nella seconda parte della sezione (2.2.1). Si ricavano innanzitutto le condizioni al contorno per l'equazione (2.3.8), chiaramente diverse rispetto al caso di input monodimensionali:

1.

$$C^{(2)}(0, p) = 0 \quad \forall p \quad (2.3.9)$$

La classificazione in un caso "degenero" di un perceptron con 0 neuroni

2.

$$C^{(2)}(n, 1) = 2 \quad \forall n > 1 \quad (2.3.10)$$

È sempre possibile trovare un iperpiano che non intersechi il segmento congiungente i punti del doppietto, il che identifica 2 dicotomie (++) e (--) ricordando che si deve avere lo stesso label per entrambi i punti del doppietto.

Nota : il caso $C(1, 1)$, $n = 1$ è non banale in quanto non avrebbe senso parlare di "distanza angolare" ρ né di "prodotto scalare" in una dimensione. Ad ogni modo si pone $C^{(2)}(1, 1) = 2 \cdot \Psi_2(\rho)$, ciò può essere motivato dal fatto che, se è possibile trovare una dicotomia identificata da un iperpiano che rispetti la proprietà di avere lo stesso label per i due punti del doppietto (la cui probabilità è $\Psi_2(\rho)$), allora esisteranno 2 possibili classificazioni per i punti del doppietto (++) e (--)).

Anche in questa circostanza si utilizza la rappresentazione a "reticolo" come in figura (2.3), cercando di rappresentare i cammini diretti che vanno da una condizione al contorno ad uno specifico $C(n, p)$ che si vuole determinare.

In questa situazione però, se si osserva l'equazione di ricorrenza (2.3.8) si notano alcune differenze rispetto al caso 1-dimensionale. Innanzitutto la base del cono di influenza è composta da $2p - 1$ termini di bordo, i quali hanno contributi che non saranno legati solo a:

1. un "movimento" verticale verso l'alto (n costante)
2. un movimento diagonale in alto e una volta verso destra ($n + 1$)

Ma, come si evince in figura (2.10), è infatti possibile anche:

3. un movimento diagonale in alto e due volte verso destra ($n + 2$)

Inoltre un'altra differenza è che i contributi dovuti a questi movimenti sono *pesati*, rispettivamente da fattori Ψ_2 , 1 e $(1 - \Psi_2)$: questo comporta il fatto che cammini diversi con gli stessi punti di partenza e arrivo, possono determinare un

2.3. PROBABILITÀ DI SEPARAZIONE PER DOPPIETTI MONODISPERSI

contributo *diverso* al valore di $C^{(2)}(n, p)$, in quanto non conta più solo il numero di cammini ma anche la "direzione media", la struttura vera e propria che compone il cammino.

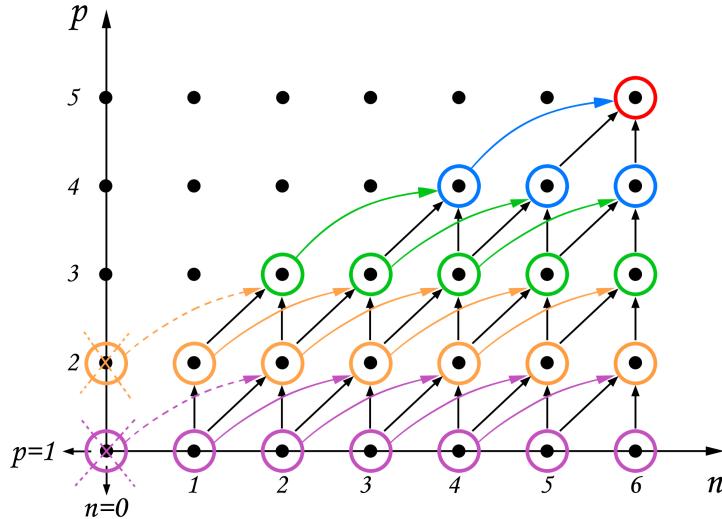


Figura 2.10: Contributi al calcolo di $C(6,5)$: dai termini di bordo rappresentati in viola, le frecce rappresentano tutti i possibili movimenti. Si nota che alcune condizioni al contorno, quelle con una freccia tratteggiata, hanno contributo nullo.

Considerando i diversi cammini e i contributi "pesati" dei termini di bordo, portando a termine i calcoli che quantificano questo ragionamento, si arriva a determinare che il contributo della i -esima condizione al contorno al valore di $C^{(2)}(n, p)$ è:

$$K(i, p) = \sum_{m=0}^{p-1} \binom{p-1}{m, i-2m} \cdot \Psi_2(\rho)^{p-1-i+m} \cdot (1 - \Psi_2(\rho))^m \quad (2.3.11)$$

Con quest'ultima relazione, esplicitando la formula di ricorrenza (2.3.8) per input a doppietti con distanza fissata, si ottiene:

$$C^{(2)}(n, p) = 2 \cdot \sum_{i=0}^{n-2} K(i, p) + 2\Psi_2(\rho) \cdot K(n-1, p) \quad (2.3.12)$$

Nel limite termodinamico, ossia per grande n , si può stimare la capacità α_c approssimando 2.3.12 con:

$$C^{(2)}(n, p) \approx 2 \cdot \sum_{i=0}^{n-2} K(i, p) \quad (2.3.13)$$

2.3. PROBABILITÀ DI SEPARAZIONE PER DOPPIETTI MONODISPERSI

Analogamente al caso di input puntiformi, α_c è il valore in cui:

$$C^{(2)}\left(\frac{p}{\alpha_c}, p\right) \approx 2^{p-1} = \frac{2^p}{2} \quad (2.3.14)$$

ossia il valore critico in cui l'apprendimento di una dicotomia da parte della rete neurale è del 50%. $n^* = \frac{p}{\alpha_c}$ è il valore di n in cui la somma $K(i, p)$ ha la metà del suo valore massimo, quando metà della base del cono di influenza è composta dai termini di bordo con contributo non nullo.

$K(i, p)$ può essere interpretato come la *funzione di partizione* di un insieme di cammini diretti $\{\gamma_j\}_{j=1,\dots,p}$ di $p - 1$ passi tali che:

- $K(i, p) \rightarrow K(n, p)$ ossia dalla i -esima condizione al contorno ad altezza $p = 1$ fino al punto finale $K(n, p)$ ad altezza p
- con movimenti possibili legati da delle "probabilità di transizione":

$$1. \text{ in verticale verso l'alto } (n \text{ costante}) \text{ con } P(\gamma_j \rightarrow \gamma_j) = \frac{\Psi_2}{2};$$

$$2. \text{ in diagonale in alto e una volta verso destra } (n + 1) \text{ con} \\ P(\gamma_j \rightarrow \gamma_j + 1) = \frac{1}{2};$$

$$3. \text{ in diagonale in alto e due volte verso destra } (n + 2) \text{ con} \\ P(\gamma_j \rightarrow \gamma_j + 2) = \frac{(1 - \Psi_2)}{2}$$

Tali probabilità sono normalizzate con fattore di normalizzazione 2 dato dalla somma dei pesi $\Psi_2 + 1 + 1 - \Psi_2 = 2$.

Per come è definita la capacità corrisponderà quindi alla *mediana* della distribuzione dei cammini così composti; approssimando con la media si ha:

$$\bar{\ell} = (p - 1) \cdot \sum_{\ell=0}^2 \ell \cdot P(\gamma_j \rightarrow \gamma_j + \ell) \quad (2.3.15)$$

dove la sommatoria a secondo fattore rappresenta lo spostamento medio verso destra che si ha ad ogni singolo passo. Sostituendo le probabilità di transizione si ottiene la relazione:

$$\bar{\ell} = (p - 1) \cdot \left(\frac{3}{2} - \Psi_2\right) \quad (2.3.16)$$

Dalla definizione di α_c , nel limite termodinamico si ottiene infine:

$$\alpha_c \approx \frac{p - 1}{\bar{\ell}} = \frac{2}{(3 - 2 \cdot \Psi_2(\rho))} \quad (2.3.17)$$

2.3. PROBABILITÀ DI SEPARAZIONE PER DOPPIETTI MONODISPERSI

Ricordando inoltre che, come in (2.3.2), $\Psi_2(\rho) = \frac{2}{\pi} \cdot \arctan \sqrt{\frac{1+\rho}{1-\rho}}$, la capacità dipende dalla distanza angolare ρ e ha il seguente andamento:

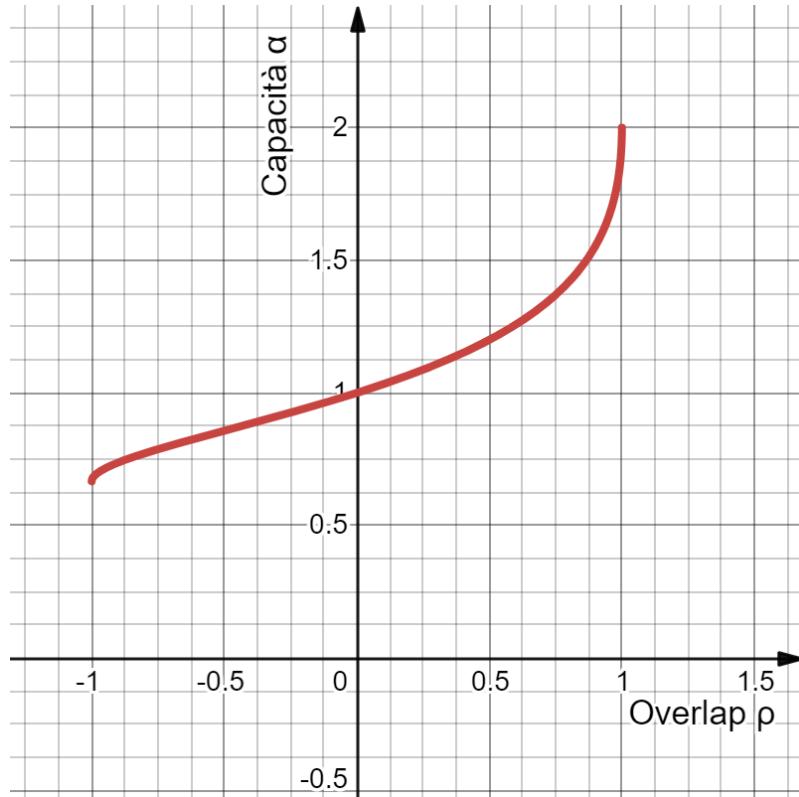


Figura 2.11: Andamento della capacità in funzione dell’overlap tra i dati strutturati in doppietti

α_c cresce al crescere di ρ : dal valore di $\alpha_c = \frac{2}{3}$ che si ha per $\rho = -1$, in modo continuo arriva fino a $\alpha_c = 2$ per $\rho = 1$.

Se l’overlap $\rho = 1$, infatti, si ricade le caso di input puntiformi:

$$\rho = \cos(\vartheta) = 1 \rightarrow \vartheta = 0 \quad (2.3.18)$$

In appendice (6.2) è presente, per completezza, l'estensione della teoria di Cover per input strutturati in multipletti di k elementi.

Capitolo 3

La programmazione lineare

La programmazione lineare è la branca della Ricerca Operativa che si occupa di studiare algoritmi di risoluzione per problemi di ottimizzazione lineare, ovvero problemi in cui sono lineari sia la funzione obiettivo (la funzione da ottimizzare), sia i vincoli a cui essa è soggetta, ovvero tutte le relazioni tra le variabili in gioco.

Già Fourier nel 1827 aveva studiato come trovare soluzioni ammissibili di un sistema di diseguaglianze lineari e, in letteratura, sono presenti lavori di von Neumann degli anni venti e trenta, sulla teoria dei giochi e su alcuni modelli economici. La paternità della programmazione lineare però viene comunemente assegnata al matematico americano George Dantzig che per primo, nel 1947 ideò un algoritmo risolutivo, il metodo del Simplex.

Per quanto l'assunzione di linearità nei fenomeni possa apparire una condizione piuttosto restrittiva, la programmazione lineare si applica ad una classe di problemi che ha un forte interesse pratico. In molte situazioni reali, infatti, i componenti di un sistema reagiscono in modo approssimativamente lineare alle decisioni prese: in questi casi sarà possibile modellare con sufficiente precisione in termini di programmazione lineare.

3.1 Modello di ottimizzazione lineare

Un problema di *linear programming* è un problema di ottimizzazione caratterizzato dalle seguenti proprietà:

- la funzione obiettivo $f(x) : \mathbb{R}^n \rightarrow \mathbb{R}$ che va massimizzata o minimizzata è lineare, ovvero della forma:

$$f(x) = c_1x_1 + \dots + c_nx_n = \sum_{j=1}^n c_jx_j \quad (3.1.1)$$

dove $\vec{c} = \{c_1, \dots, c_n\}$ sono i coefficienti delle funzione obiettivo e $\vec{x} = \{x_1, \dots, x_n\}$ sono dette variabili decisionali;

- i vincoli del problema, equazioni/disequazioni lineari presenti in numero finito m , sono relazioni che legano tra loro le variabili decisionali, ad esempio:

$$a_1x_1 + \dots + a_nx_n \leq b_i \text{ con } i = 1, \dots, m \quad (3.1.2)$$

dove $\vec{a} = \{a_1, \dots, a_n\}$ sono detti coefficienti tecnologici e $\vec{b} = \{b_1, \dots, b_m\}$ sono i termini noti dei vincoli (o risorse).

Un problema lineare corrisponde quindi ad un sistema lineare

$$\begin{cases} \max f(x) = c_1x_1 + \dots + c_nx_n \\ a_{1,1}x_1 + \dots + a_{1,n}x_n \leq b_1 \\ \vdots \\ a_{m,1}x_1 + \dots + a_{m,n}x_n \leq b_m \end{cases} \quad (3.1.3)$$

In una forma più compatta, si ha

$$\begin{cases} \max f(x) = \vec{c} \cdot \vec{x} \\ A \cdot \vec{x} \leq \vec{b} \end{cases} \quad (3.1.4)$$

dove \vec{c} e \vec{x} sono vettori n -dimensionali, \vec{b} è un vettore m -dimensionale e A è la matrice $m \times n$ dei coefficienti tecnologici, $\begin{pmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \cdots & a_{m,n} \end{pmatrix}$

3.1. MODELLO DI OTTIMIZZAZIONE LINEARE

Ciò, si esprime in *forma canonica* come:

$$\begin{array}{ll} \text{Massimizza} & \vec{c}^T \cdot \vec{x} \\ \text{soggetto ai vincoli} & A \cdot \vec{x} \leq \vec{b} \\ \text{e} & \vec{x} \geq 0 \end{array}$$

dove per completezza, si aggiunge un vincolo di *non* negatività delle variabili decisionali.

Lo spazio delle soluzioni ammissibili è un *politopo* convesso ottenuto dall'intersezione finita di "sottospazi", ognuno definito da una disequazione di vincolo. La funzione obiettivo è definita su di esso e un algoritmo risolutivo di linear programming trova, se esiste, un punto del politopo in corrispondenza del quale la funzione obiettivo assume il suo valore minimo o massimo.

3.1.1 Dualità dei problemi lineari

Un'importantissima proprietà dei problemi lineari è la cosiddetta *dualità*, ovvero assegnato un problema di programmazione lineare (P1) detto primale, ad esso è possibile associare un altro problema (P2) detto duale, le cui soluzioni saranno strettamente legate a quelle del primo. Il duale è formulato da informazioni contenute nel problema originale e, quando risolto, esso fornisce informazioni essenziali alla soluzione del primale.

P1)		P2)	
<i>Minimizza</i>	$\vec{c}^T \cdot \vec{x}$	<i>Massimizza</i>	$\vec{b}^T \cdot \vec{y}$
<i>soggetto ai vincoli</i>	$A \cdot \vec{x} \geq \vec{b}$	<i>soggetto ai vincoli</i>	$A \cdot \vec{y} \leq \vec{c}$
e	$\vec{x} \geq 0$	e	$\vec{y} \geq 0$

(dove $\vec{c}, \vec{x} \in \mathbb{R}^n$; $\vec{y} \in \mathbb{R}^m$; $A \in \mathbb{R}^{m \times n}$ e $\vec{b} \in \mathbb{R}^m$)

È immediato osservare che mentre il primale è un problema di minimo, il duale è un problema di massimo e oltre ai versi opposti delle disequazioni di vincolo i termini noti di un problema sono i coefficienti della funzione obiettivo dell'altro.

Studio come sono legate le soluzioni di problema primale e problema duale.

Sia x^* una soluzione del problema P1; allora per ogni $y \in \mathbb{R}^m$, $y \geq 0$, poiché $A \cdot x^* \geq b$ risulta:

$$c^T x^* \geq c^T x^* + y^T (b - Ax^*) \quad (3.1.5)$$

$$\geq \min_{x \in \mathbb{R}^n} c^T x + y^T (b - Ax) \quad (3.1.6)$$

$$= b^T y + \min_{x \in \mathbb{R}^n} (c^T - y^T A)x \quad (3.1.7)$$

Affinchè il problema di minimizzazione presente nel secondo membro di quest'ultima relazione (3.1.7) non sia illimitato inferiormente, è necessario che il vettore y sia tale che:

$$A^T y = c \quad (3.1.8)$$

Infatti se così non fosse e risultasse $A^T y \neq c$ allora si potrebbe scegliere $x(\theta) = -\theta(A^T y - c)$ in corrispondenza del quale il problema di minimizzazione diventa:

$$\min_{\theta} -\theta \|A^T y - c\|^2 \quad (3.1.9)$$

che risulti illimitato inferiormente, ovvero che P1 sia senza soluzione (al contrario di quanto ipotizzato).

Nell'ipotesi $A^T y = c$, il membro destro di (3.1.7) si riduce a $b^T y$, risulterà quindi:

$$c^T x^* \geq b^T y \quad (3.1.10)$$

Per rendere il più possibile stringente la stima della limitazione inferiore del valore ottimo della funzione obiettivo $c^T x^*$ si può rendere quanto più grande possibile il termine di destra della diseguaglianza (3.1.10), cioè si può massimizzare la quantità $b^T y$ al variare del vettore $y \in \mathbb{R}^m$, tra tutti i vettori che soddisfano $A^T y = c$, $y \geq 0$. Più formalmente si ha:

$$c^T x^* \geq b^T y^* \quad (3.1.11)$$

dove y^* è soluzione del problema seguente:

$$\begin{array}{ll} \text{Massimizza} & \vec{b}^T \cdot \vec{y} \\ \text{soggetto ai vincoli} & A \cdot \vec{y} \leq \vec{c} \\ \text{e} & \vec{y} \geq 0 \end{array}$$

Quest'ultimo prende il nome di problema *duale* (P2); inoltre è possibile dimostrare che, se l'insieme delle soluzioni di P1 è *non* vuoto e P1 ha soluzione finita x^* , allora esisterà un y^* soluzione del problema duale tale che:

$$c^T x^* = b^T y^* \quad (3.1.12)$$

ossia che i valori delle funzioni obiettivo per le due soluzioni (del problema primale e del duale) coincidono.

3.2 Problemi continui: Algoritmo del simplexso

La principale classe di problemi lineari, e anche quella di maggior interesse per le applicazioni riguardanti questo lavoro di tesi è quella dei problemi cosiddetti *continui*, dove le variabili decisionali \vec{x} possono assumere con continuità tutti i valori contenuti all'interno del loro dominio di esistenza.

Il principale algoritmo di risoluzione è il *metodo del Simplexso*. Dal nome si evince chiaramente che esso fa uso del concetto di simplexso, ossia un politopo di $n + 1$ vertici in n dimensioni, ad esempio un segmento in una dimensione, un triangolo in 2 dimensioni, un tetraedro in 3 dimensioni... e così via. Si è di fronte ad un problema lineare così formulato:

$$\text{Massimizza} \quad \vec{c}^T \cdot \vec{x} \quad (3.2.1)$$

$$\text{soggetto ai vincoli} \quad A \cdot \vec{x} \leq \vec{b} \quad (3.2.2)$$

$$e \quad \vec{x} \geq 0 \quad (3.2.3)$$

Come detto precedentemente l'insieme dei punti che soddisfano tutti i vincoli del problema definiscono un politopo non vuoto, limitato o illimitato, esso prende il nome di regione *ammissibile* in cui si possono verificare le seguenti situazioni:

- Il problema non ammette soluzione in quanto:
 - la regione ammissibile è vuota;
 - la regione ammissibile è illimitata e la funzione obiettivo è illimitata superiormente (se il problema è di massimizzazione) o illimitata inferiormente (se il problema è di minimizzazione).
- la regione ammissibile è limitata: in questo caso il problema ammette soluzione, detta *ottima*.

L'algoritmo del simplexso è in grado di determinare il tipo di politopo definito dai vincoli e di trovare la soluzione ottima valutando i valori della funzione obiettivo sui punti del politopo. Si può infatti dimostrare che se il massimo della funzione

obiettivo è assunto nella regione ammissibile, allora tale valore si avrà in un vertice del politopo che delimita tale regione di spazio. Se su un vertice non viene assunto il massimo/minimo, si può verificare che esiste un bordo del politopo in cui la funzione obiettivo cresce/decresce strettamente allontanandomi da esso.

Se il bordo è finito è quindi collegato ad un altro vertice dove potrà essere assunto il valore estremante; in caso contrario, se il bordo non è finito, la funzione obiettivo risulterà illimitata e il problema lineare non ammetterà soluzione.

L'algoritmo si "muove" quindi da un vertice all'altro del politopo in modo contiguo verso un valore più alto (o più basso) della funzione obiettivo, in base a se si vuole massimizzare (o minimizzare). Questo procedimento è iterato fino al raggiungimento del max/min o se si effettua un "movimento" verso un bordo illimitato. Tale sequenza di operazioni termina sempre, in quanto il numero di vertici di un politopo è sempre un valore finito; il tempo di computazione però risulta variabile perché strettamente dipendente dal numero di vertici *visitati*.

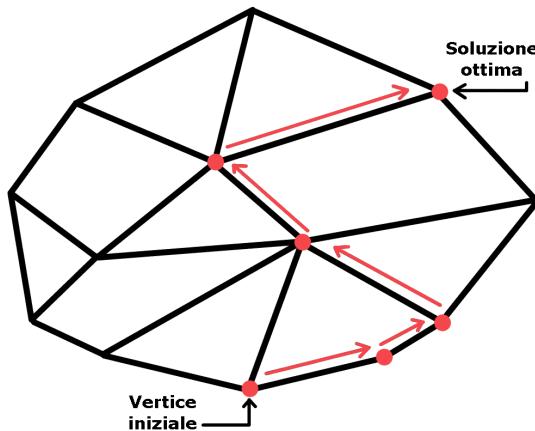


Figura 3.1: Sequenza di iterazioni attraverso i vertici del politopo, per arrivare alla soluzione ottima

L'algoritmo si applica a problemi formulati in forma standard, ovvero in cui i vincoli sono rappresentati in forma di uguaglianze, ad esempio:

1. vincoli della forma : $x_i \leq n$

si introduce una variabile di *slack* s per cui il vincolo diventa: $x_i + s = n$

2. vincoli della forma : $x_i \geq n$

si introduce una variabile di *surplus* s per cui il vincolo diventa: $x_i - s = n$

In questo modo i vincoli presenti in (3.2.2) diventeranno della forma $A \cdot \vec{x} = \vec{b}$ dove la matrice A conterrà anche delle entrate relative alle nuove variabili di slack e

surplus introdotte (lo stesso accadrà per il vettore \vec{c}^T dei coefficienti della funzione obiettivo).

Il problema si può equivalentemente esprimere in forma tabellare:

$$\begin{array}{lll} \text{Massimizza} & \vec{c}^T \cdot \vec{x} \\ \text{soggetto ai vincoli} & A \cdot \vec{x} \geq \vec{b} & \leftrightarrow \\ e & \vec{x} \geq 0 & \begin{pmatrix} 1 & -\vec{c}^T & 0 \\ 0 & A & \vec{b} \end{pmatrix} \end{array}$$

La prima riga definisce la funzione obiettivo, le altre righe i vincoli; il vettore nullo della seconda riga ha componenti in numero pari a quelle di \vec{b} , ovvero m .

L'algoritmo del simplex procede come segue:

1. Verifica di ottimalità

Condizione sufficiente affinchè la tabella sia ottima è che:

- le entrate $-\vec{c}^T$ siano tutte *non positive* ≤ 0 se si sta minimizzando
- le entrate $-\vec{c}^T$ siano tutte *non negative* ≤ 0 se si sta massimizzando

2. Se la tabella non è ottima si sceglie una colonna h corrispondente al massimo (minimo) valore tra le entrate di $-\vec{c}^T$ se si sta minimizzando (massimizzando). In entrambi i casi ce ne sarà almeno uno, altrimenti l'algoritmo si sarebbe fermato al punto 1.

3. Verifica di illimitatezza

Condizione sufficiente affinchè il problema sia *illimitato* è che nella colonna h scelta si abbiano solo valori negativi nelle entrate della matrice A , cioè che

$$a_{i,h} < 0 \quad \forall i = 1, \dots, m \quad (3.2.4)$$

In tal caso il problema è illimitato lungo questa direzione

4. Se non si è in presenza di un problema illimitato si sceglie il *pivot*, ovvero l'entrata della tabella che genera il minimo rapporto tra il termine noto (ultima colonna a destra) e il coefficiente della relativa variabile nella colonna h della matrice A ; l'indice di riga di questa entrata sarà quindi:

$$i' = \min \left\{ \frac{b_i}{a_{i,h}} : a_{i,h} > 0, \quad \forall i = 1, \dots, m \right\} \quad (3.2.5)$$

Si applica quindi la cosiddetta operazione di cardine, ovvero si effettua un "movimento" lungo una direzione ammissibile per un certo passo, in modo

che la soluzione sia anch'essa ammissibile e, al tempo stesso, migliore della precedente.

L'algoritmo procede poi ricorsivamente fino alla soluzione ottimale, fino al vertice del politopo rappresentante la regione ammissibile in cui la funzione obiettivo assume il suo valore estremale. In appendice (6.3), è presente un esempio di applicazione del metodo del simplex.

3.3 Problemi interi: Algoritmo Branch and Bound

Per quanto riguarda i problemi *interi*, l'algoritmo principale di risoluzione è il Branch and Bound; sviluppato da A.H. Land e A.G. Doig nel 1960, esso si basa sulla scomposizione del problema originale in sottoproblemi più semplici da risolvere.

Si suppone di avere un problema P^0 :

$$P^0 = (z, F(P^0)) \quad (3.3.1)$$

dove z è la funzione obiettivo che si vuole massimizzare/minimizzare e $F(P^0)$ è la regione ammissibile.

La soluzione *ottima* sarà

$$z^* = z(P^0) = \{z(x) : x \in F(P^0)\} \quad (3.3.2)$$

mentre z^{best} rappresenta la miglior soluzione ammissibile *nota a priori*. Si procede nel suddividere P^0 in k sottoproblemi P^1, P^2, \dots, P^k la cui totalità rappresenta P^0 , e $F(P^0)$ in k sottoinsiemi $\{F(P^1), F(P^2), \dots, F(P^k)\}$ tali che $\bigcup_{i=1}^k F(P^i) = F(P^0)$. Preferibilmente tali sottoregioni di ammissibilità sono tra loro *disgiunte*, ovvero

$$F(P^i) \cap F(P^j) = \emptyset \quad \forall i \neq j \quad (3.3.3)$$

La scomposizione in sottoproblemi si può anche iterare, arrivando così ad avere un albero decisionale:

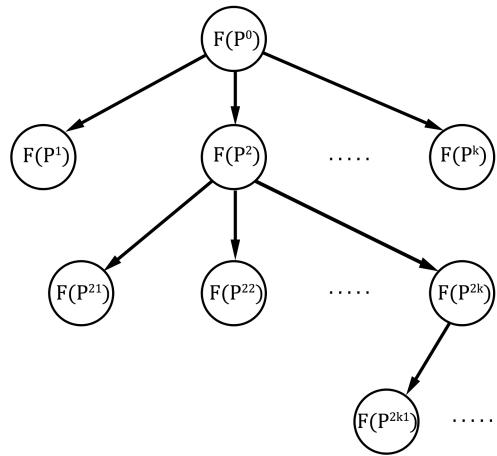


Figura 3.2: Ogni freccia rappresenta la relazione di discendenza. Il nome dell'algoritmo deriva proprio dal processo di *ramificazione*, in inglese appunto, branching

Risolvere il problema P^0 è equivalente al risolvere tutti i P^k sottoproblemi generati, immaginando ad esempio di voler minimizzare, si avrà:

$$z^* = z(P^0) = \min\{z(P^1), z(P^2), \dots, z(P^k)\} \quad (3.3.4)$$

Un sottoproblema P^i si può considerare risolto se:

1. Si determina la soluzione ottima di P^i ;
 2. Si dimostra che la sottoregione $F(P^i)$ è vuota, ovvero che P^i non ha soluzione;
 3. Si dimostra che $z(P^i) \geq z^{best}$ ($z(P^i) \leq z^{best}$ se si sta massimizzando), ovvero che la soluzione del sottoproblema è *peggiore* della migliore conosciuta.

Se, invece, non si riesce a risolvere un nodo, si suddivide anch'esso in ulteriori sottoproblemi e si itera la ricerca di una soluzione. Per ogni sottoproblema P^i è possibile determinare un Lower Bound L della soluzione in modo da seguire una strategia di esplorazione dell'albero più efficiente:

$$L(P^i) \leq z(P^i) \quad (3.3.5)$$

cioè la più piccola soluzione al problema P^i .

Se $L(P^i) \geq z^{best}$, il nodo si può escludere, in quanto la miglior soluzione che si può sperare di ottenere è peggiore della soluzione ammissibile del problema.

Per ottenere un Lower Bound di $P^i = (z, F(P^i))$, si deve trovare uno dei cosiddetti *rilassamenti* del problema, $R(P^i) = (z_r, F_r(P^i))$ tale che:

- $F_r(P^i) \supseteq F(P^i)$;
- $z_r(y) \leq z(y) \quad \forall y \in F(P^i)$

Il problema rilassato è risolvibile in modo più semplice rispetto al problema originale, è quindi possibile trovarne la soluzione ottima, che rappresenta il Lower Bound del problema originale; tale rilassamento infatti è scelto in modo da essere il più "vicino" possibile al problema originale.

L'algoritmo quindi separa ricorsivamente lo spazio ammissibile in sottospazi più piccoli, in ognuno di essi procede a minimizzare/massimizzare la funzione obiettivo (qui si è in grado di farlo), per poi effettuare un confronto tra i valori ottenuti e la soluzione z^{best} nota a priori. In questo ultimo step dell'algoritmo, che verrà iterato per tutti i sottoproblemi, può avvenire:

Nel caso di **minimizzazione**:

- se $z(P^i) \geq z^{best}$
la soluzione del sottoproblema viene scartata
- se $z(P^i) < z^{best}$
 $z^{best} = z(P^i)$
ovvero la nuova soluzione migliore diventa la soluzione che ha "superato" il confronto

Nel caso di **massimizzazione**:

- se $z(P^i) \leq z^{best}$
la soluzione del sottoproblema viene scartata
- se $z(P^i) > z^{best}$
 $z^{best} = z(P^i)$
ovvero la nuova soluzione migliore diventa la soluzione che ha "superato" il confronto

Questi due casi presentati sono uno il problema *Duale* dell'altro.

In appendice (6.4) è presente un esempio di applicazione del metodo Branch and Bound. L'algoritmo risolutivo di questa sezione è stato presentato per completezza, in quanto esso non potrà essere utilizzato: nell'analisi effettuata sul perceptron le variabili presenti sono infatti continue.

Capitolo 4

Simulazioni numeriche e risultati ottenuti

Attraverso metodi numerici che confrontino le formule teoriche con le misurazione effettuate su un perceptron, si valuta se la teoria esposta nel capitolo 2 sia rispettata sia per input senza struttura, sia per input a doppietti. Normalmente per eseguire le simulazioni si utilizza il *Perceptron Algorithm*, che però ha il difetto di essere lento a livello di tempistiche computazionali; per superare questo ostacolo, essendo il perceptron un separatore dicotomico *lineare* si è pensato per le simulazioni numeriche di utilizzare tecniche ed algoritmi di *Linear Programming*. A tal proposito in questo capitolo si introducono le librerie utilizzate e si mostrerà l'algoritmo vero e proprio, analizzandone le parti più importanti.

4.1 Misura numerica della probabilità di separazione con metodi di linear programming

Il motore di base scritto in linguaggio C++ ha lo scopo di valutare l'apprendimento medio di un perceptron su un certo set di input e di label, che vengono generati attraverso un motore random inizializzato secondo le condizioni che si desidera verificare. In questo elaborato si è scelto di utilizzare un metodo di linear programming, contenuto nelle librerie open-source di Google, gli **OR-Tools** [5].

Esse comprendono gli algoritmi per risolvere problemi di *ottimizzazione combinatoria*, ovvero in grado di trovare la miglior soluzione (detta *ottima*) in un ampio insieme di soluzioni possibili. Le librerie OR-Tools includono vari risolutori, per problemi di programmazione a vincoli, per vehicle routing ed in particolare, per

4.1. MISURA NUMERICA DELLA PROBABILITÀ DI SEPARAZIONE CON METODI DI LINEAR PROGRAMMING

il nostro utilizzo, metodi di programmazione lineare. Nello specifico si utilizzerà il Glop Linear Solver, efficiente a livello di memoria e numericamente stabile; il Google linear programming system, che formalizza il già citato metodo del Simplex, trova il valore ottimo di una funzione obiettivo soggetta ad un set di vincoli lineari.

Come visto precedentemente infatti, in problemi di questo tipo gli ingredienti fondamentali sono:

- La funzione obiettivo, ossia la quantità da ottimizzare;
- I vincoli, le restrizioni presenti sul set delle possibili soluzioni, legati specificatamente alle richieste del problema.

Nel nostro caso specifico caso di studio, la funzione obiettivo avrà come variabili il vettore n -dimensionale dei pesi \vec{w} e sarà rappresentata da:

$$\vec{w} \cdot \vec{0} \quad (4.1.1)$$

ovvero si lavorerà con una funzione già ottimizzata (avendo valore costante sempre nullo), essendo più nello specifico interessati a quando tale vettore \vec{w} soddisfa contemporaneamente tutti i vincoli del problema, espressi dalla funzione modello della rete neurale:

$$sgn(\vec{w} \cdot \vec{\xi^i}) = sgn(\sigma_i) \quad (4.1.2)$$

dove $\vec{\xi^i}$ è l' i -esimo vettore dei p input e σ_i è il label corrispondente.

La finalità del programma sarà quindi quella di effettuare una statistica sul numero di volte in cui è restituita la soluzione ottima, ovvero le volte in cui l'apprendimento di una dicotomia da parte del perceptron è avvenuta con successo.

4.1.1 Funzionamento dell'algoritmo

Lo scopo della simulazione è confrontare ad n fissato la probabilità di apprendimento di una dicotomia da parte di un perceptron al variare di p , con il valore previsto secondo la formula teorica.

Si passa ora a descrivere nel dettaglio il funzionamento del programma di simulazione, scritto in linguaggio C++ .

4.1. MISURA NUMERICA DELLA PROBABILITÀ DI SEPARAZIONE CON METODI DI LINEAR PROGRAMMING

Parametri fissi

- p : il numero dei vettori di input che si andrà ad indagare;
- n : il numero di neuroni con cui verranno costruite le reti neurali all'interno della simulazione, corrispondente al numero di dimensioni dello spazio delle fasi;
- $N_{iteraz} = 1000$: il numero ripetizioni che il programma effettua, a fissati valori di p e di n , alla ricerca di una soluzione ottima per i valori delle componenti del vettore dei pesi \vec{w}
- $N = 50$: il numero di misure che verranno effettuate per ogni valore di n e di p per poi poter effettuare una statistica.

Alcune di queste variabili vengono inizializzate esplicitamente nel codice sorgente, pertanto il loro valore viene attribuito in *compile time*. Ciò ha come conseguenza che se si vuole modificarne il valore, occorre modificare il codice sorgente e ricompilare l'eseguibile. Si è scelto di optare per questa soluzione in modo che ogni eseguibile costituisse un diverso "esperimento": ogni simulazione effettuata dallo stesso eseguibile mantiene la stessa struttura del perceptron testato e la stessa statistica. Questo è importante perché saranno confrontate diverse simulazioni effettuate con gli stessi parametri a livello di eseguibile, ma si avranno eseguibili differenti per verificare che le conclusioni tratte da un certo set-up siano valide anche in altre condizioni.

Il primo passaggio è quello di generare casualmente i vettori del training set e i rispettivi label. Utilizzando i costrutti Random della *standard library* del C++ si settano i generatori di input casuali e la distribuzione gaussiana da cui saranno estratti i vettori $\vec{\xi}^i$.

```
1 random_device rd{};
2 mt19937 gen{rd()};
3 normal_distribution<double> distrib{0.0 , 1.0};
```

4.1. MISURA NUMERICA DELLA PROBABILITÀ DI SEPARAZIONE CON METODI DI LINEAR PROGRAMMING

Per la generazione dei label σ_i si costruisce una funzione che restituisce casualmente uno tra i valori +1 o -1:

```
1 int RandomBool();
2 int RandomBool(){
3     int x = rand();
4
5     if (x > (RAND_MAX/2)) {
6         return 1;
7     } else {
8         return -1;
9    }
```

I passaggi principali nella risoluzione del problema utilizzando gli OR-Tools di Google sono:

1. Includere le librerie specifiche contenenti l'algoritmo che si vuole utilizzare;

```
1 #include "ortools/linear_solver/linear_solver.h"
```

2. Dichiaraone del *solver*, nel nostro caso, trattandosi di programmazione lineare, esso sarà il GLOP;

```
1 int LinearProgramming(int n_input) {
2
3     MPSolver solver("linear_programming",
4                     MPSolver::GLOP_LINEAR_PROGRAMMING);
5
6     const double infinity = solver.infinity();
```

La funzione LinearProgramming ha come argomento un intero pari al numero di input della rete neurale e restituisce un intero. Il solver della libreria utilizzata ha inoltre bisogno della definizione di infinito.

3. Creazione delle variabili della funzione obiettivo. Nel nostro caso esse sono le componenti del vettore dei pesi \vec{w} , per ognuna delle n componenti va specificato il range di esistenza, nel nostro caso tra $-\infty$ e $+\infty$ e il nome assegnatogli;

4.1. MISURA NUMERICA DELLA PROBABILITÀ DI SEPARAZIONE CON METODI DI LINEAR PROGRAMMING

```

1 vector <string> name(n);
2     for (int j=0; j<n ; j++){
3         name[j] = "w" + to_string(j);
4     }; //fisso il nome di ognuna delle n componenti del vettore pesi w
5
6 vector <MPVariable*> w(n); //creazione del vettore pesi n dimensionale
7
8     for (int j=0; j<n; j++){ //setto le componenti
9         w[j] = solver.MakeNumVar(-infinity, infinity, name[j]);
10    };

```

4. Definizione dei vincoli riguardanti le variabili create. Si avranno p vincoli, ognuno rappresentato dalla funzione forward della rete neurale, proprio come in (4.1.2); i metodi presenti nella libreria prevedono che per ogni vincolo definito, siano specificati i coefficienti di ogni variabile interessata ($\{w_i\}_{i=1,\dots,n}$) e il "range" del vincolo. Quest'ultimo è legato ai due argomenti richiesti dal metodo MakeRowConstraint(): due numeri, il primo rappresentante l'estremo inferiore e il secondo l'estremo superiore presenti nel vincolo (ad esempio se un vincolo fosse $x + y \geq 3$ si avrebbe MakeRowConstraint(3.0 , infinity) e se invece fosse $x + y \leq -4$ si avrebbe MakeRowConstraint(-infinity , -4.0)).

I vincoli del perceptron sono in particolare:

$$\begin{cases} \vec{w} \cdot \vec{\xi^i} > 0 & \text{se } \sigma_i = +1 \\ \vec{w} \cdot \vec{\xi^i} < 0 & \text{se } \sigma_i = -1 \end{cases} \quad (4.1.3)$$

In forma più compatta si possono esprimere i vincoli come:

$$\sigma_i \cdot (\vec{w} \cdot \vec{\xi^i}) > 0 \quad (4.1.4)$$

4.1. MISURA NUMERICA DELLA PROBABILITÀ DI SEPARAZIONE CON METODI DI LINEAR PROGRAMMING

```

1 vector<int> label(n_input); //definizione dei p label di input
2 srand(time(0)); //seed per il RandomBool
3
4 vector < MPConstraint* > vinc(n_input);
5
6 for (int i=0; i<n_input; i++){
7     label[i] = RandomBool(); //label i-esimo
8     vinc[i] = solver.MakeRowConstraint(Epsilon, infinity);
9         //creazione del vincolo i-esimo
10
11    for (int j=0; j<n; j++){
12        //definizione del vincolo i-esimo specificando ogni coefficiente
13            // delle componenti del vettore pesi
14        vinc[i]->SetCoefficient( w[j] , label[i] * distrib(gen) );
15    };
16 };

```

Dalla relazione (4.1.4) si evince che il coefficiente per ogni componente del vettore \vec{w} è dato dal prodotto del label i -esimo con un numero generato casualmente secondo la distribuzione gaussiana (`distrib(gen)`).

Si è però di fronte ad un problema: i vincoli in programmazione lineare sono sempre rappresentati con disequazioni aventi simboli \leq e \geq , nel nostro caso però, nei vincoli (4.1.4) è presente il simbolo di > 0 , l'uguaglianza non è inclusa.

Si è osservato che il programma, se come bound inferiore del vincolo si utilizza il numero 0.0, trova sempre una soluzione ottima in corrispondenza di tutte le componenti w_i nulle. Questo caso è ovviamente da escludere, inserendo come primo argomento del metodo un numero *Epsilon* appena superiore a 0; in seguito si ragionerà su quanto questo numero influisca nei confronti con i risultati teorici.

5. Definizione della funzione obiettivo da ottimizzare. Come detto precedentemente si è scelta la funzione (4.1.1), dove ognuna delle n variabili w_i create nel punto 3), avrà coefficiente nullo.

4.1. MISURA NUMERICA DELLA PROBABILITÀ DI SEPARAZIONE CON METODI DI LINEAR PROGRAMMING

```
1 MPOjective* const objective = solver.MutableObjective();  
2  
3     for (int j=0; j<n; j++){  
4         objective->SetCoefficient( w[j] , 0.);  
5     };  
6  
7     objective->SetMaximization(); //parte l'ottimizzazione
```

6. Invocazione del solver per ottenere l'eventuale soluzione ottimale. La libreria possiede nativamente un modo per comunicare un insuccesso, ovvero se il problema inserito non ha soluzione ottima o è illimitato. Se così accade l'apprendimento del perceptron avrà esito negativo.

```
1 const MPSolver::ResultStatus result_status = solver.Solve();  
2 // Controllo se il problema ha una soluzione ottima  
3 if (result_status != MPSolver::OPTIMAL) {  
4     ns++; //esito Negativo : incremento del contatore del numero di insuccessi  
5 } else{  
6     s++; //esito Positivo : incremento del contatore del numero di successi  
7 }  
8  
9 return s ;  
10 }
```

La funzione iniziale LinearProgramming restituisce il numero di successi, ovvero quante volte l'apprendimento del perceptron è avvenuto con successo.

Questo programma, fissato un valore di n e di p , sarà iterato N_{iteraz} volte e si studierà la statistica degli esiti positivi. N_{iteraz} è fissato come detto in precedenza a 1000, e il rapporto $\frac{\text{successi}}{N_{\text{iteraz}}}$ rappresenterà la probabilità che il perceptron apprenda una data classificazione, ovvero quante delle 1000 dicotomie di input sono linearmente separabili.

Si ripete questa iterazione tenendo n fissato e facendo variare p , per osservarne la curva di apprendimento ed effettuare un confronto con la curva teorica ottenuta secondo i calcoli di Cover del capitolo 2.

4.1. MISURA NUMERICA DELLA PROBABILITÀ DI SEPARAZIONE CON METODI DI LINEAR PROGRAMMING

4.1.2 Risultati per input non strutturati

Si sono analizzati i casi con n fissato (3 , 5 , 6 , 10) e p variabile da 0 a 20. Come nel grafico teorico, sulle ascisse si ha il $load = \frac{p}{n}$ mentre sulle ordinate si ha la probabilità di apprendimento.

Nel caso teorico essa è definita come da formula (2.1.2) come rapporto delle dicotomie linearmente separabili sulle dicotomie totali $\frac{C(n,p)}{2^p}$.

Nelle simulazioni numeriche invece, si effettuano N_iteraz 1000 iterazioni a n fissato per ogni valore di p dell'intervallo, ognuna alla ricerca della soluzione ottimale per il vettore dei pesi \vec{w} . Come ultimo passaggio il programma scrive in un file di testo l'output della simulazione $\left(\frac{\text{successi}}{N_iteraz} \right)$, ovvero la probabilità di apprendimento da parte della rete neurale.

Il tutto viene ripetuto $N= 50$ volte, per ogni n fissato e per ogni p dell'intervallo scelto, ottenendo così 50 curve di apprendimento per ogni valore scelto del numero di neuroni n .

Si procede poi ad un'analisi dei dati: si calcola una curva media, mediando le 50 probabilità di apprendimento per ogni valore di p e si effettua il confronto con la curva teorica rappresentando il tutto in un grafico.

Per riprodurre la curva teorica è stato sviluppato un programma in linguaggio **Python**, il quale utilizza i risultati delle formule teoriche di Cover sugli stessi valori di n scelti e sui medesimi intervalli di variazione di p . In questa circostanza si è preferito cambiare linguaggio in quanto **Python** ha un miglior comportamento nella gestione di numeri molto alti: il range di esistenza delle variabili intere, infatti, è molto più ampio rispetto a quello di **C++**.

L'obiettivo delle simulazioni è verificare che i dati misurati siano compatibili con le previsioni date dal modello teorico, occorre verificare che il valore di $Prob_{teo}(n, p)$ sia contenuto entro le barre d'errore di $Prob_{mis}(n, p)$. Si considera il livello di confidenza al 99.7 %, ossia 3σ .

4.1. MISURA NUMERICA DELLA PROBABILITÀ DI SEPARAZIONE CON METODI DI LINEAR PROGRAMMING

Si considerano quindi due grandezze, la discrepanza e la deviazione standard della media di una misura di probabilità di apprendimento.

$$\Delta_p = |Prob_{mis} - Prob_{teo}| \quad (4.1.5)$$

$$\hat{\sigma}_p = \sqrt{\frac{\sum_{i=1}^N (Prob_{mis}^i - < Prob_{mis}^i >)^2}{N(N-1)}} \quad (4.1.6)$$

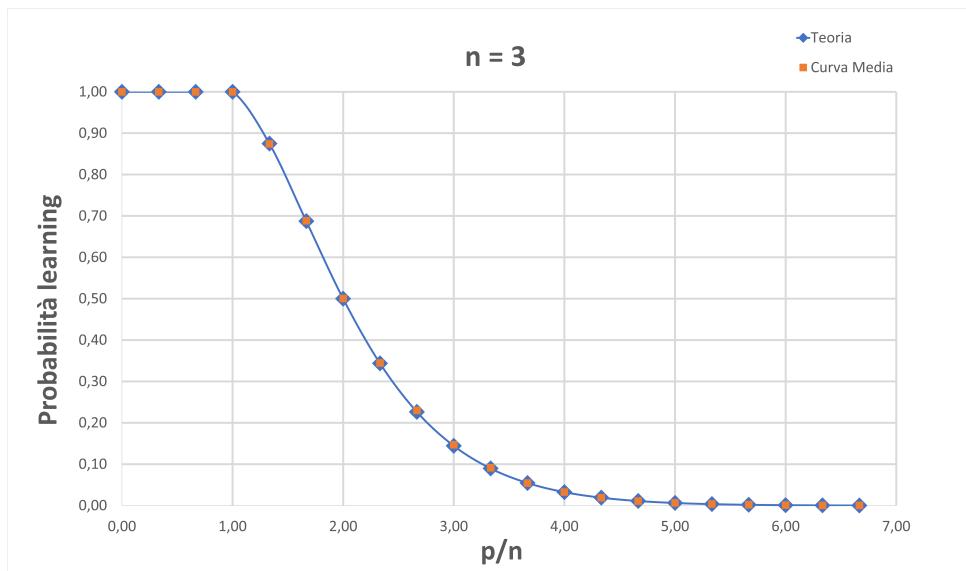
dove $N = 50$ rappresenta il numero di misure effettuate per ogni valore di p , presente in pedice. Essendo lo scopo delle simulazioni comparare l'adeguatezza della teoria alle misure per ogni valore di p , sono necessari degli estimatori di discrepanza ed incertezza statistica relativi all'intera simulazione. Si definiscono dunque:

$$\Delta = \sqrt{\sum_{p=1}^{p_{max}} \Delta_p^2} \quad (4.1.7)$$

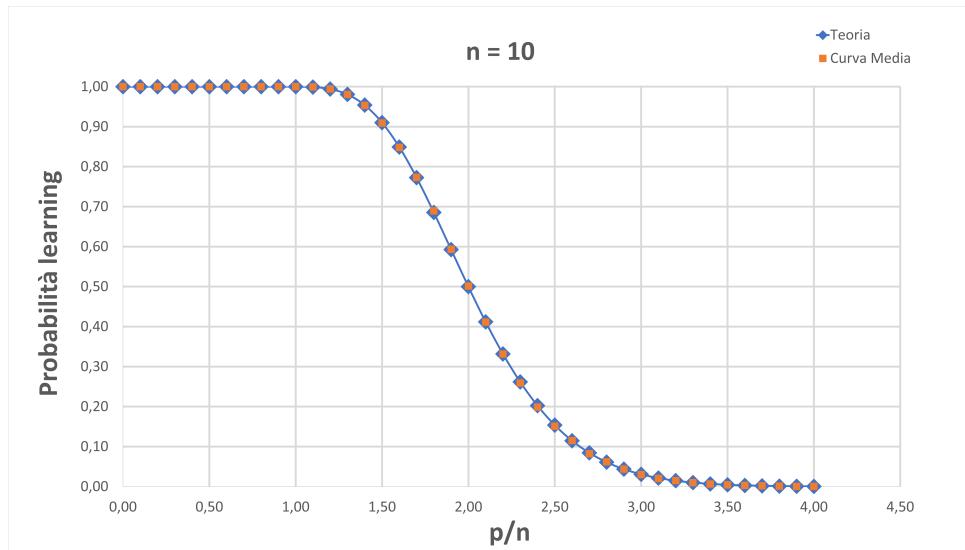
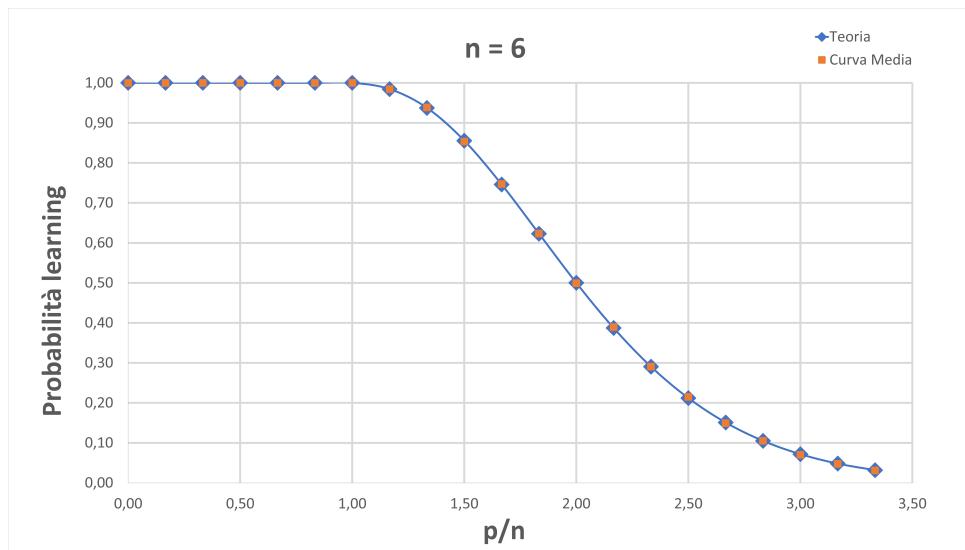
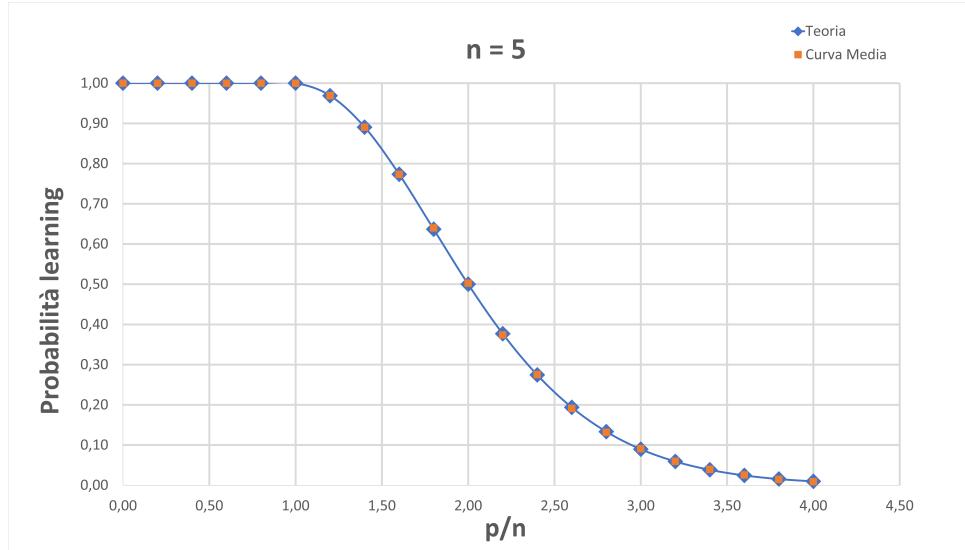
$$\hat{\sigma} = \sqrt{\sum_{p=1}^{p_{max}} \hat{\sigma}_p^2} \quad (4.1.8)$$

In questo modo, una singola misura della probabilità di *learning* sarà considerata valida se $\Delta_p < 3\sigma_p$, mentre l'intera simulazione sarà valida se $\Delta < 3\sigma$

Di seguito sono riportati i grafici relativi alle curve di apprendimento per input puntiformi, sperimentali e teoriche, per $n = 3, 5, 6, 10$.



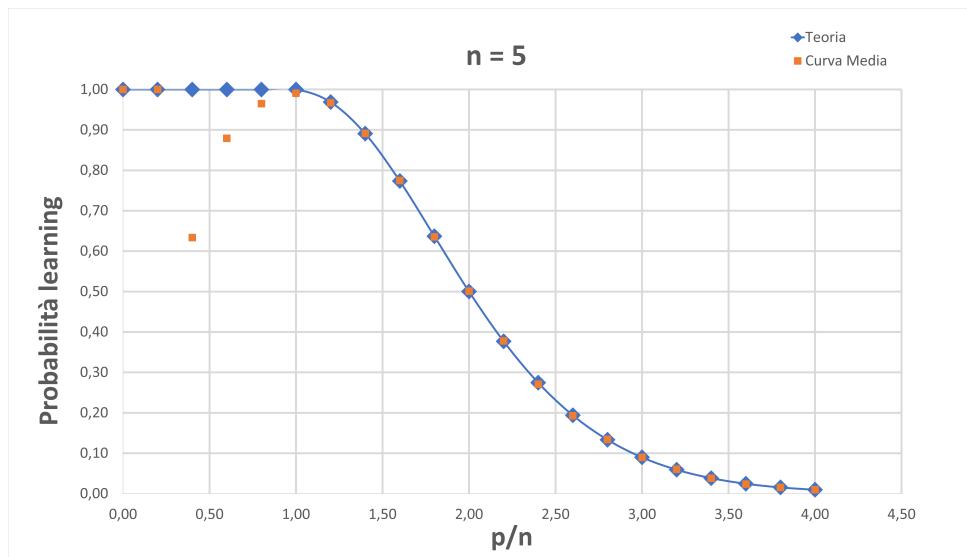
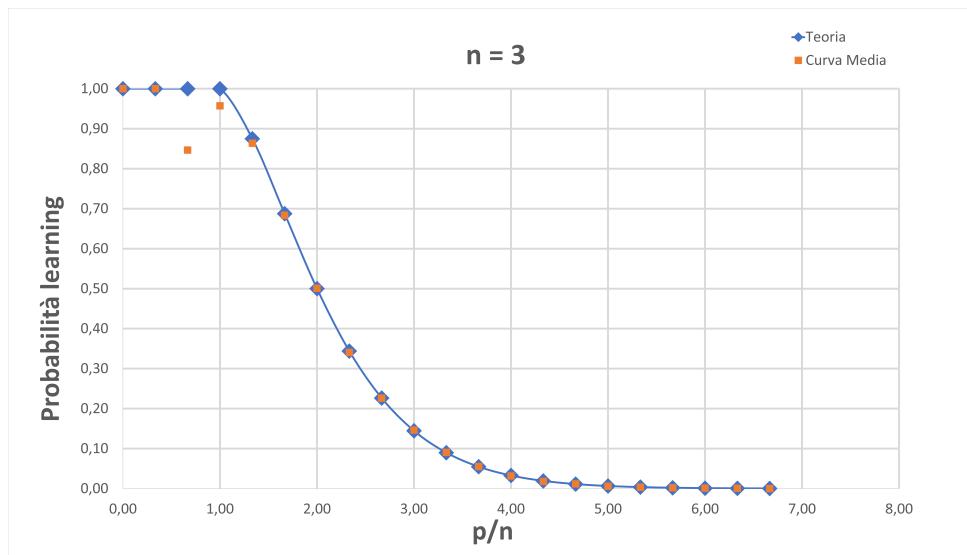
4.1. MISURA NUMERICA DELLA PROBABILITÀ DI SEPARAZIONE CON METODI DI LINEAR PROGRAMMING

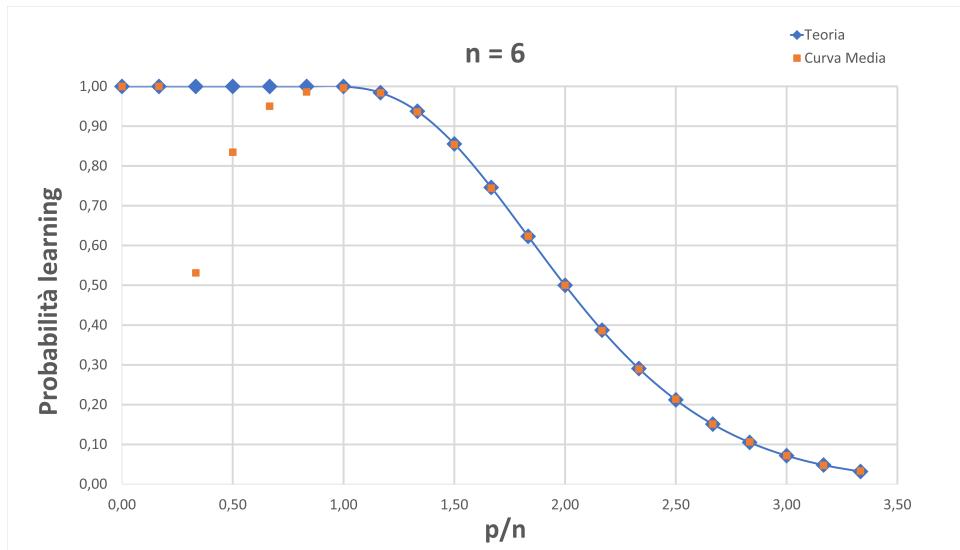


4.1. MISURA NUMERICA DELLA PROBABILITÀ DI SEPARAZIONE CON METODI DI LINEAR PROGRAMMING

Si evince chiaramente che le simulazioni numeriche rispettano pienamente la teoria: le barre d'errore sono presenti nel grafico ma, essendo relative alla terza/quarta cifra decimale, non risultano nemmeno visibili. Si possono quindi riscontrare le conferme delle considerazioni esposte nel capitolo 2, ossia che tutte le dicotomie sono linearmente realizzabili se $p \leq n$ (la Probabilità di learning è del 100%) e che la capacità (la condizione in cui l'apprendimento è possibile per metà delle dicotomie) si ha quando $p = 2n$.

Si sono inoltre effettuate le stesse simulazioni cambiando il parametro *Epsilon* del codice sorgente. I primi grafici riportati hanno questo parametro fissato a 1×10^{-11} ; studiando il comportamento per diversi valori di Epsilon, si è notato che quando tale parametro è alzato fino a 1×10^{-4} si ottengono risultati diversi, che non rispettano pienamente la teoria:





Il perceptron ha infatti un improvviso calo di apprendimento nella fase iniziale, per poi tornare progressivamente a rispettare la curva teorica con un'ottima compatibilità tra i dati. Tale comportamento si è riscontrato per tutti i valori di n analizzati; si è quindi deciso di fissare il parametro Epsilon al valore di 1×10^{-11} per la fase di simulazione successiva, dove si analizzano dati strutturati a doppietti.

4.2 Algoritmo per input a doppietti

Nell'analisi del caso di input a doppietti il codice sorgente per le simulazioni ha tantissime analogie con il caso di input non strutturati: il Solver della libreria e la funzione obiettivo saranno gli stessi, mentre i cambiamenti principali riguarderanno il Training Set e, di conseguenza, la definizione dei vincoli.

Come nel caso precedente come prima cosa si settano i generatori di input casuali e la distribuzione gaussiana da cui saranno estratti i vettori ξ^i e $\bar{\xi}^i$.

```

1 random_device rd{};
2 mt19937 gen{rd()};
3 normal_distribution<double> distrib{0.0 , 1.0};
4 normal_distribution<double> distrib_dopp {0.0 , sigma}; //generator partner
    
```

La seconda distribuzione prevede un parametro fissato σ , il cui significato verrà approfondito in seguito.

4.2. ALGORITMO PER INPUT A DOPPIETTI

Si definiscono poi alcune funzioni che saranno utilizzate nella generazione del Training set:

```
1 template <typename Container>
2
3 double norm_of(const Container & v){ //funzione norma
4     return sqrt( accumulate(v.begin(), v.end(), 0., [](double acc, double s)
5                     { return acc + s*s; } ) );
6 }
7
8 void normalize (vector<double> & v){ //funzione di normalizzazione
9     double norm = norm_of(v);
10    for (auto & elem:v) { elem /= norm;}
11 }
```

Per la generazione dei vettori $\{\xi^i \text{ e } \bar{\xi}^i\}$ si implementano le seguenti funzioni:

```
1 vector<double> spherepoint(int dimensione) {
2     vector<double> point(dimensione);
3
4     for(int i=0; i<dimensione; i++) {
5         point[i]= distrib(gen);
6     };
7     normalize(point);
8
9     return point;
10};
```

Questa prima genera un vettore di dimensione pari a quella specificate nell'argomento settando le varie componenti secondo dei numeri casuali secondo una gaussiana; prima del return viene effettuata una normalizzazione del vettore generato.

```

1 vector<double> partner(const vector<double> & v ){
2
3     vector<double> dopp(v.size());
4
5     for (int i=0; i < v.size(); i++){
6
7         dopp[i]= v[i] + distrib_dopp(gen);
8     };
9     normalize(dopp);
10
11    return dopp;
12 }

```

Per quanto riguarda $\bar{\xi^i}$, gli elementi correlati del doppietto, essi sono generati a partire dal vettore passato in argomento (ξ^i) a cui sarà sommato un numero estratto da una seconda distribuzione gaussiana caratterizzata dal parametro sigma. Anche in questo caso prima del return viene effettuata una normalizzazione del vettore generato.

Si passa quindi alla generazione vera e propria del Training Set: a n e p fissati (quest'ultimo è rappresentato dal parametro `n_input` passato come argomento della funzione `LinearProgramming`):

```

1 vector<vector<double>> vettinp(n_input , vector<double> (n) ); //xi
2 vector<vector<double>> dopp_inp(n_input , vector<double>(n) ); //xi-barra
3
4 for (int t=0; t < n_input ; t++){
5     vettinp[t] = spherepoint(n);
6     dopp_inp[t] = partner(vettinp[t]);
7 };
8
9 for (int l=0; l<n_input; l++){
10 prodScal += inner_product( vettinp[l].begin(), vettinp[l].end(), dopp_inp[l].begin(), 0. );
11 };
12 overlap = ( prodScal )/n_input;

```

si generano quindi p vettori n -dimensionali e i loro p partner correlati, anch'essi ovviamente n dimensionali, e se ne calcola l'overlap medio ρ secondo la formula (2.3.1).

4.2. ALGORITMO PER INPUT A DOPPIETTI

Il passaggio successivo è quello di definizione dei vincoli sulle variabili della funzione obiettivo (i pesi w_i): in totale si avranno $2p$ vincoli, p sui vettori ξ e p sui vettori $\bar{\xi}$.

```

1 vector<int> label(n_input);
2 srand(time(0));
3
4 vector < MPCConstraint* > vinc(2*(n_input)); //2p vincoli
5
6 for (int i=0; i<(n_input); i++){
7     label[i] = RBool();
8     vinc[i] = solver.MakeRowConstraint(Epsilon, infinity); //vincoli xi
9     vinc[i+n_input] = solver.MakeRowConstraint(Epsilon, infinity); //vincoli xi-barra
10
11    for (int j=0; j<n; j++){
12        vinc[i]->SetCoefficient( w[j] , label[i] * vettinp[i][j] );
13        vinc[i + n_input]->SetCoefficient( w[j] , label[i] * dopp_inp[i][j] );
14    };
15}

```

Ogni vincolo avrà come buond inferiore il numero Epsilon, proprio come nel caso di dati non strutturati.

La più grande differenza rispetto al caso analizzato in precedenza è a livello di setting dei coefficienti delle variabili della funzione obiettivo: bisogna tenere conto delle proprietà dei doppietti, in quanto i due elementi ξ^i e $\bar{\xi}^i$ hanno lo stesso label, in quanto dovranno essere classificati dal perceptron allo stesso modo.

Per rendere funzionale il programma il tipo di ritorno della funzione LinearProgramming si è scelto di tipo *pair* (un container della libreria standard del C++), in cui il primo elemento conterrà il numero di successi da parte del perceptron a livello di apprendimento e il secondo l'overlap medio della simulazione.

```

1 pair <int , double > LinearProgramming(int n_input)

```

Anche in questo caso, fissato un valore di n e di p il programma sarà iterato N_{iteraz} volte e si studierà la statistica degli esiti positivi. N_{iteraz} è fissato come detto in precedenza a 1000, e il rapporto $\frac{successi}{N_{iteraz}}$ rappresenterà la probabilità che il perceptron apprenda una data classificazione, ovvero quante delle 1000 dicotomie di input sono linearmente separabili.

Si ripete questa iterazione tenendo n fissato e facendo variare p , per osservarne la curva di apprendimento ed effettuarne un confronto con quella teorica ottenuta secondo i calcoli di Cover del capitolo 2.

Per ogni simulazione si calcola inoltre l'overlap medio, parametro che caratterizza la disposizione dei doppietti e la curva teorica. Tale valore di distanza angolare ρ è strettamente dipendente dal parametro *sigma* presente nel codice: si varierà quest'ultimo per ottenere simulazioni riguardanti dati con diversi valori di ρ tra i punti del doppietto.

4.2.1 Risultati per input a doppietti

Si sono analizzati i casi con n fissato (3 , 6 , 10 , 20) e p variabile. Come nel grafico teorico, sulle ascisse si ha il $load = \frac{p}{n}$ mentre sulle ordinate si ha la probabilità di apprendimento.

Nelle simulazioni numeriche si effettuano N_iteraz 1000 iterazioni a n fissato per ogni valore di p dell'intervallo scelto, ognuna alla ricerca della soluzione ottimale per il vettore dei pesi \vec{w} . Come ultimo passaggio il programma scrive in un file di testo l'output della simulazione $\left(\frac{\text{successi}}{N_iteraz} \right)$ e l'overlap medio $\langle \rho \rangle$ tra i doppietti analizzati.

Variando il parametro *sigma* si sono ottenuti doppietti con diversi valori di distanza angolare $\langle \rho \rangle$: per ogni "esperimento" ad n fissato si sono quindi analizzati dati, sfruttando l'approssimazione di *mean field*, con

$$\langle \rho \rangle \approx 0, 0.2, 0.5, 0.8, 1 \quad (4.2.1)$$

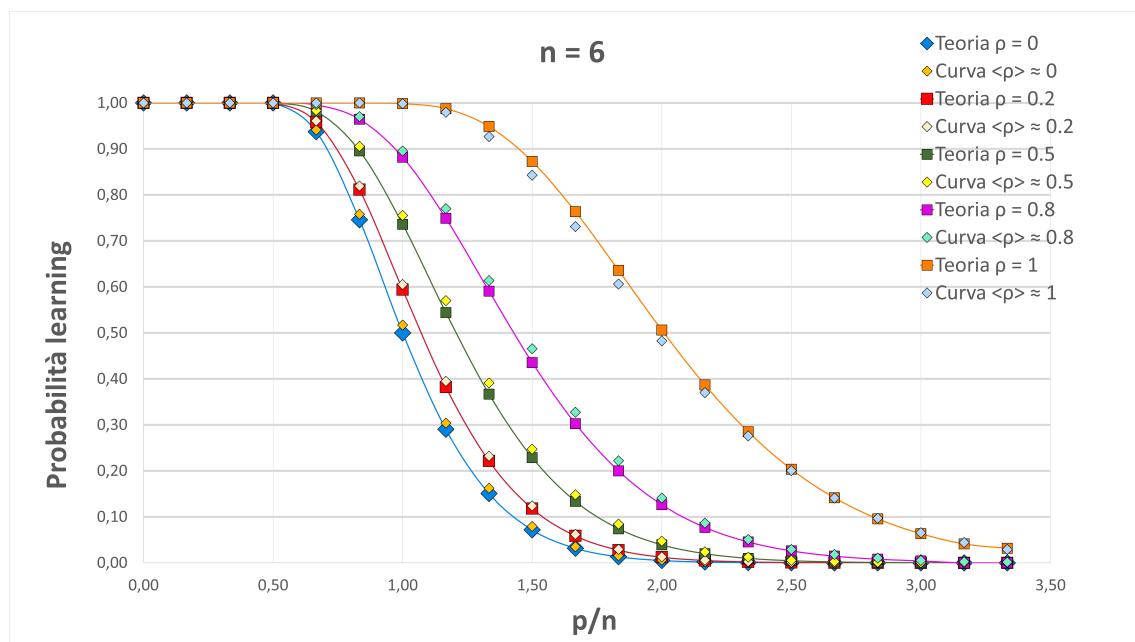
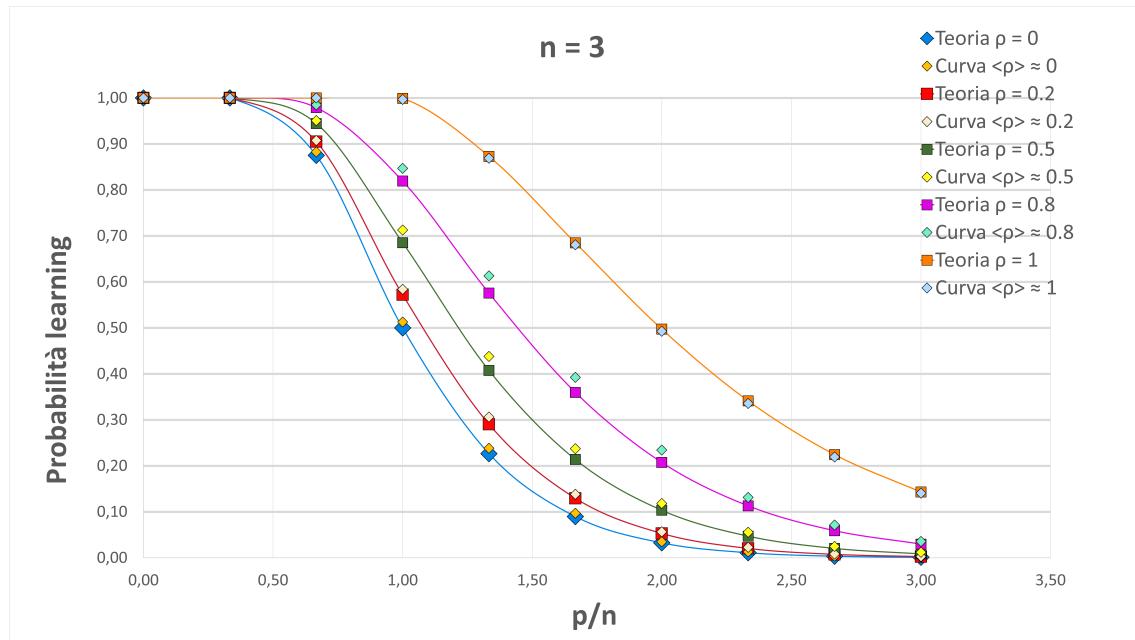
Il tutto viene ripetuto $N= 50$ volte, per ogni n fissato e per ogni p , ottenendo così 50 curve di apprendimento per ogni valore scelto del numero di neuroni n e di distanza angolare $\langle \rho \rangle$.

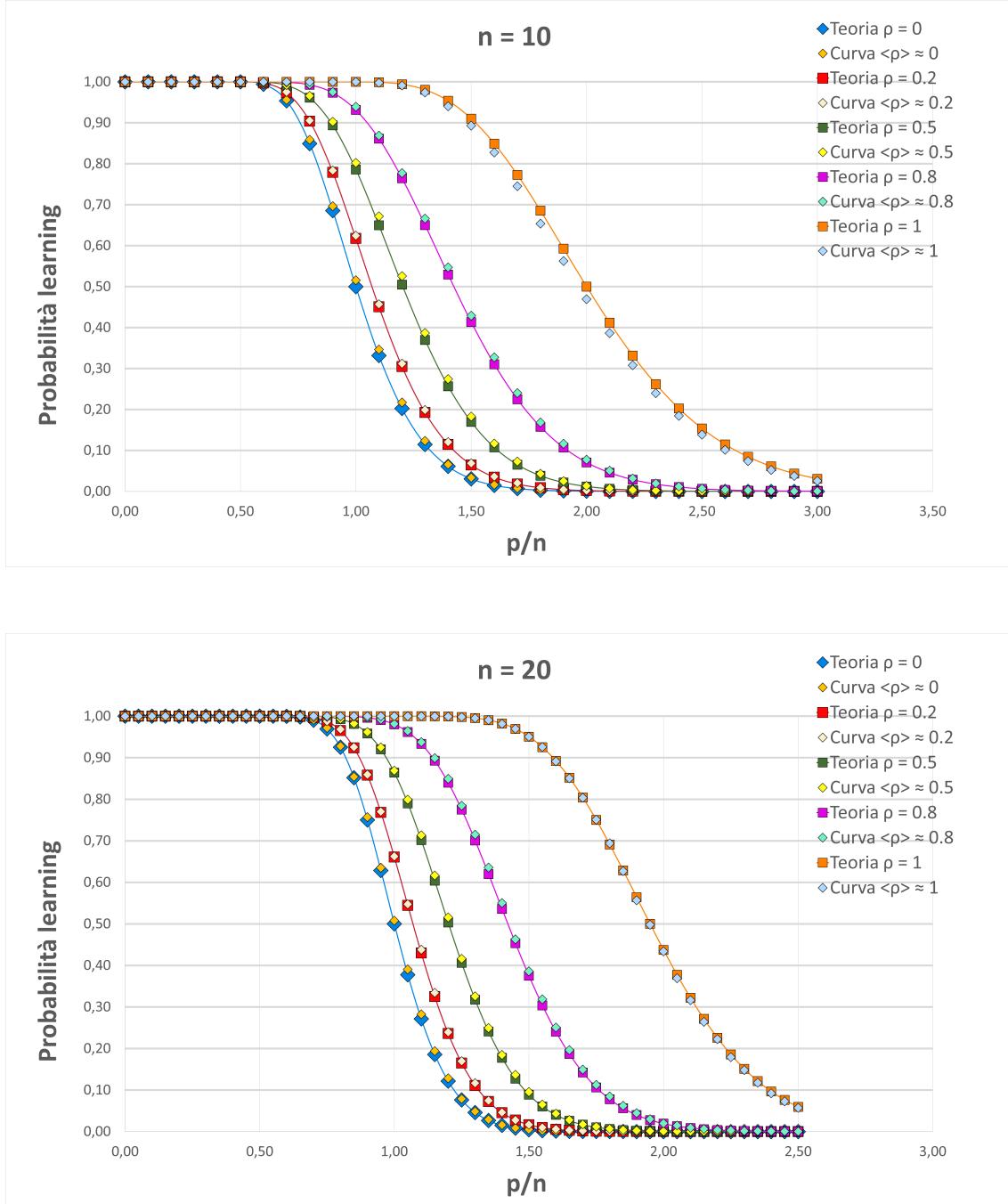
Come nel caso di input puntiformi si calcola una curva media su 50 probabilità di apprendimento per ogni valore di p e si effettua il confronto con le predizioni teoriche mediante un grafico. In questa circostanza per riprodurre la curva teorica è stato sviluppato un algoritmo in linguaggio **Mathematica**, il quale riproduce la formula di ricorrenza per i doppietti (2.3.8) sugli stessi valori di n scelti e sui medesimi intervalli di variazione di p , tenendo anche conto dei ragionamenti sulle condizioni al contorno discussi nella nota 2.3.3.

Per le discrepanze e l'incertezze statistiche, si è utilizzato il medesimo ragionamento degli input puntiformi.

4.2. ALGORITMO PER INPUT A DOPPIETTI

Di seguito sono riportati i grafici.





Si evince che c'è un'ottima compatibilità tra le simulazioni numeriche e le curve teoriche: le barre d'errore sono presenti nel grafico ma, essendo relative alla terza/quarta cifra decimale, non risultano nemmeno visibili. Mediamente inoltre, le incertezze statistiche diminuiscono all'aumentare del numero di dimensioni n ma aumentano all'aumentare del valore di $\langle p \rangle$.

Si notano infatti alcune inadeguatezze, soprattutto quando $\rho \gtrsim 0,7$: per tali valori $\Psi_2(\rho)$ è in un regime non lineare. Nell'equazione (2.3.8) si nota che il numero di dicotomie linearmente separabili $C(n, p, \rho)$ dipende dall'overlap soltanto attraverso

4.2. ALGORITMO PER INPUT A DOPPIETTI

$\Psi_2(\rho)$ e proprio per questo motivo è in tale regime che si accentua la differenza $\langle \Psi_2(\rho) \rangle \neq \Psi_2(\langle \rho \rangle)$ causando alcune mancanze nell'approssimazione di mean field.

La differenza tra valori sperimentali e teorici è proprio da imputare a questo fatto: si considera un solo valore di ρ calcolandone il valore medio e un solo valore della grandezza $\Psi_2(\rho)$ per tutta la simulazione; la variazione alla grandezza $\Psi_2(\rho)$ causata dalla diversità tra gli overlap dei doppietti analizzati, così facendo, non è considerata.

Capitolo 5

Conclusioni

Le reti neurali sono uno strumento molto utilizzato nella società moderna; sul loro sviluppo ci sono grandi investimenti da parte della ricerca ma anche del business. I vantaggi applicativi possibili sono tantissimi e proprio per questo c'è stata una grande accelerazione per quanto ne riguarda lo studio ingegneristico, volto ad ampliarne sempre di più le potenzialità. Il rapido sviluppo tecnologico ha comportato un'ampia disparità tra ciò che viene realizzato e la teoria quantitativa che lo descrive. Da parte della comunità scientifica, tra cui fisici statistici e fisici dei sistemi complessi, risulta quindi di grande interesse cercare di colmare questo divario, arricchendo l'impianto teorico in modo che esso sia più adatto alle necessità attuali.

La teoria analitica sviluppata finora infatti è in grado di descrivere quantitativamente il comportamento di reti neurali con basso livello di complessità, sotto le ipotesi che i dati siano distribuiti uniformemente nello spazio delle fasi e che siano indipendenti gli uni dagli altri. Alcuni recenti studi cercano di rendere meno rigida quest'ultima ipotesi, in modo da avvicinarsi a casi reali in cui i dati possiedono una certa struttura. Vengono così definiti i *manifold*, agglomerati di punti che possiedono caratteristiche comuni. I manifold sono strutture molto generiche, a cui possono essere date svariate specializzazioni, ad esempio quella di *politopo*, un manifold geometricamente costruito in modo che i punti appartenenti abbiano distanza reciproca fissata. Nel caso di studio di un *perceptron*, ossia di un classificatore lineare, la distanza che si considera è l'overlap. È stata elaborata una teoria che misura l'apprendimento di un perceptron nel caso in cui gli input siano strutturati in politopi aventi tutti lo stesso overlap. Questo elaborato focalizza l'attenzione sul valore della frazione di dicotomie che

il perceptron è in grado di apprendere su set di dati parametrizzati da: p che corrisponde al numero di input ricevuti, n le dimensioni dello spazio ambiente e la distribuzione con cui viene estratto l'overlap ρ . Tale frazione di dicotomie dove le separazioni sono dette linearmente separabili è di grande interesse, in quanto è una quantità che rappresenta la probabilità che un set di input strutturato secondo certe caratteristiche venga appreso da un perceptron; una sorta di misura della "propensione all'apprendimento" di questa rete neurale.

Si confrontano le misure di tale grandezza con le predizioni teoriche e ne viene studiato il comportamento al variare dei parametri che definiscono l'input, utilizzando algoritmi di linear programming. In primo luogo le simulazioni vengono effettuate in un contesto per cui esiste una teoria già convalidata. In questa fase l'algoritmo di Linear Programming ha dimostrato di essere estremamente affidabile, sia quando le simulazioni si svolgono in dimensione ridotta ($n = 3$ e $n = 5$) sia quando si aumenta la complessità algoritmica ($n = 10$). Sfruttare le tecniche di programmazione lineare inoltre si è rivelata la scelta giusta anche a livello di tempistiche computazionali le quali rappresentavano uno dei maggiori ostacoli e una fonte d'errore per quanto riguarda l'algoritmo più comunemente usato in letteratura, il Perceptron Algorithm.

La compatibilità tra le misure sperimentali e teoriche è ottima anche nella seconda fase di simulazione dove si è verificata l'ipotesi di *mean field*, misurando che l'apprendimento della rete neurale su un set di input polidisperso è uguale a quella che si avrebbe su un set di input monodisperso dove la distanza angolare corrisponde alla media delle distanze tra i dati.

I risultati ottenuti dimostrano come ci si attendeva la validità del mean field per grandi valori di n ($n = 10$ e $n = 20$); anche "lontani" dal limite termodinamico, ossia in regimi di dimensioni ridotte ($n = 3$ e $n = 6$) però si ha un'ottima compatibilità tra teoria e risultati numerici. In generale non vi è motivo per cui questa ipotesi debba invalidarsi a certi valori di n , poiché nessun vincolo è stato posto a questo parametro né per la formulazione teorica né per la costruzione dell'algoritmo. Proprio per questa ragione si può concludere che l'approssimazione di mean field sia efficace in qualsiasi spazio delle fasi con dimensione finita.

Molti sono ancora gli aspetti da approfondire e soprattutto i possibili ulteriori sviluppi: la verifica della validità del *mean field* anche per input strutturati in multipletti di overlap variabile rappresenterebbe infatti una situazione molto più vicina alla realtà rispetto ai doppietti studiati in questa tesi. Un altro grande

risultato sarebbe lo sviluppo di una teoria riguardante il comportamento del perceptron nel caso in cui venga meno l'ipotesi di isotropia della posizione degli input. Come detto in precedenza, infatti, lo studio effettuato riguarda set di input con posizione totalmente casuale sulla sfera unitaria nello spazio delle fasi. Nell'ipotesi in cui il dominio dei possibili posizionamenti dei punti sia ristretto oppure se si rendesse più probabile il posizionamento in una certa zona piuttosto che in un'altra, i risultati rappresenterebbero situazioni specifiche che si potrebbero avere in un caso più tangibile.

In ambito di reti neurali tanti sono gli orizzonti aperti su cui si può proseguire ad investigare, ad esempio migliorarne sempre di più le caratteristiche e l'adattabilità ampliandone le potenzialità che esse hanno nelle applicazioni pratiche. Si può essere certi però che i ruoli futuri delle reti neurali saranno strettamente legati a quanto bene l'uomo riuscirà a comprenderle.

Proprio come all'inizio della tesi si è tornati ad uno dei paradigmi fondamentali della vita umana: l'apprendimento.

Capitolo 6

Appendice

6.1 Calcolo di $\Psi_2(\rho)$

La grandezza $\Psi_2(\rho)$ quantifica la frazione di iperpiani nello spazio che assegnano lo stesso label ad entrambi i punti costituenti un doppietto $\{\xi, \bar{\xi}\}$; essa può essere riscritta come:

$$\Psi_2(\rho) = \frac{2}{\aleph} \int d^n x \delta(|x|^2 - 1) \cdot \theta(x \cdot \xi_x) \cdot \theta(x \cdot \bar{\xi}_x) \quad (6.1.1)$$

dove il fattore di normalizzazione $\aleph = \int d^n x \delta(|x|^2 - 1) = \frac{\Omega_n}{2}$, con Ω_n l'angolo solido in n dimensioni.

$\Psi_2(\rho)$ è un integrale n -dimensionale con proprietà di simmetria sferica ($n - 2$)-dimensionale; è quindi possibile effettuare un cambiamento di coordinate attraverso una trasformazione ortogonale, verso una nuova base Y scelta opportunatamente. Mediante l'ortonormalizzazione di Gram-Schmidt si ha che:

$$\begin{cases} e_1 = \xi_y \\ e_2 = \frac{\bar{\xi}_y - \rho \xi_y}{\sqrt{1 - \rho^2}} \end{cases} \rightarrow \begin{cases} \xi_y = e_1 \\ \bar{\xi}_y = \rho e_1 + (\sqrt{1 - \rho^2}) e_2 \end{cases}$$

Dove $\rho = \xi \cdot \bar{\xi}$ è l'overlap, la distanza angolare tra i due vettori del doppietto.

La definizione di prodotto scalare è indipendente dalla scelta del sistema di coordinate; essendo la nuova base scelta in modo tale che ci si trovi nel piano parallelo a ξ e $\bar{\xi}$, tutti gli altri $(n - 2)$ piani siano ortogonali a questo e, di conseguenza, i termini presenti nell'integrale (6.1.1) $(\vec{y} \cdot \vec{\xi})$ e $(\vec{y} \cdot \vec{\bar{\xi}})$ restituiranno solamente due degli n termini del prodotto scalare in quanto i restanti sono nulli.

L'integrale si riduce quindi ad un integrale in due dimensioni che, utilizzando le coordinate polari:

$$\begin{cases} e_1 = r \cdot \cos(\varphi) \\ e_2 = r \cdot \sin(\varphi) \end{cases} \quad (6.1.2)$$

Diventa:

$$\Psi_2(\rho) = \int \frac{d\Omega_2}{\pi} \theta(\cos(\varphi)) \cdot \theta(\rho \cos(\varphi) + \sqrt{1 - \rho^2} \sin(\varphi)) \quad (6.1.3)$$

Sviluppando i calcoli si arriva al risultato finale, ovvero:

$$\boxed{\Psi_2(\rho) = \frac{2}{\pi} \cdot \arctan \sqrt{\frac{1+\rho}{1-\rho}}} \quad (6.1.4)$$

6.2 Teoria di Cover per politopi generici (k -upletti)

Dati strutturati in multipletti di k punti $\xi^\mu = \{\xi_1^\mu, \dots, \xi_p^\mu\}$ con $\mu = 1, \dots, k$. Il Training Set sarà composto da p k -upletti ognuno formato da vettori n dimensionali. Una dicotomia è linearmente separabile (e quindi realizzabile da un perceptron) se tutti i k punti del k -upletto sono classificati allo stesso modo, ovvero la rete neurale assegna loro lo stesso output:

$$f(\xi_i^\mu) = f(\xi_i^\nu) \quad \forall i = 1, \dots, p \quad \text{e} \quad \mu, \nu = 1, \dots, k \quad (6.2.1)$$

Si estende la teoria di Cover ai k -upletti, ovvero si cerca la formula di ricorrenza estendendo il ragionamento effettuato nel capitolo 2 per il caso di doppietti a distanza fissata.

Allo step $(p+1)$ considero il k -upletto $\xi_{p+1} = \{\xi_{p+1}^1, \dots, \xi_{p+1}^k\}$, escludo momentaneamente ξ_{p+1}^1 e applico il Function Counting Theorem sui $(k-1)$ punti "rimasti" del set:

$$\overline{\xi_{p+1}} = \{\xi_{p+1}^2, \dots, \xi_{p+1}^k\} \subset \xi_{p+1} \quad (6.2.2)$$

(avendo un punto in meno sarà chiaramente contenuto nel multipletto iniziale).

Otterrò quindi un'espressione per il numero di dicotomie:

$$Q^{(k-1)}(C^{(k)}(n, p), C^{(k)}(n-1, p), \dots, C^{(k)}(n-k+1, p)) \quad (6.2.3)$$

Tale numero sarà funzione di:

- $C^{(k)}(n, p)$: numero di dicotomie su k -upletti senza vincoli sulle n dimensioni; identificate cioè da iperpiani non passanti per nessuno dei k punti;
- $C^{(k)}(n - 1, p)$: numero di dicotomie su k -upletti con **1** vincolo sulle n dimensioni; identificate cioè da iperpiani passanti per **1** dei k punti;
- $C^{(k)}(n - k + 1, p)$: numero di dicotomie su k -upletti con **k-1** vincoli sulle n dimensioni; identificate cioè da iperpiani passanti per **k-1** dei k punti;

In generale tale numero di dicotomie $Q^{(k-1)}$ sarà funzione di

$$C^{(k)}(n - \ell, p) \text{ con } \ell = 1, \dots, k - 1 \quad (6.2.4)$$

In ogni passaggio della teoria di Cover:

1. Parto da un numero di dicotomie linearmente separabili realizzate su uno specifico numero di punti;
2. Aggiungo un ulteriore punto, conseguenza: *variazione* sul numero di dicotomie linearmente separabili rappresentata dalle dicotomie che sono identificate con un iperpiano passante per il punto aggiunto. Ovvero tale variazione sarà rappresentata dalle dicotomie linearmente separabili con un vincolo sulle dimensioni dello spazio ambiente (imponendo il passaggio per il "nuovo" punto).

Se ho un set di $(k - 1)$ punti avrò:

- 1 termine "senza vincoli dimensionali"
- $(k - 1)$ termini ognuno dei quali deve sottostare a vincoli dimensionali dello spazio ambiente sempre maggiori:

$$C^{(k)}(n - 1, p), C^{(k)}(n - 2, p), \dots, C^{(k)}(n - k + 1, p)$$

3. In ogni aggiunta di un punto bisognerà poi tenere conto oltre ai vincoli dimensionali e ai passaggi degli iperpiani sui punti aggiunti, anche delle relazioni tra i punti a livello di struttura dei dati.

Si avranno perciò anche "modifiche" sul numero di dicotomie dovute ai vincoli di uguaglianza tra i label di tutti i punti del multipletto: le dicotomie che assegnano al punto aggiunto un label diverso da quelli degli altri punti del multipletto saranno infatti da scartare.

La quantità $Q^{(k-1)}$ rappresenta il numero di dicotomie linearmente separabili, realizzate dal set:

$$\{\xi_1 \cup \xi_2 \cup \dots \cup \xi_p\} \quad (6.2.5)$$

p multipletti di k -elementi, dove vale $f(\xi_i^\mu) = f(\xi_i^\nu)$

$\forall i = 1, \dots, p$ e $\mu, \nu = 1, \dots, k$, cioè dove i k elementi dell' i -esimo k -upletto hanno tutti lo stesso label.

A cui è unito $\overline{\xi_{p+1}}$, il multipletto $(p+1)$ -esimo di $(k-1)$ elementi (gli si è tolto un elemento), dove vale $f(\xi_{p+1}^\mu) = f(\xi_{p+1}^\nu) \quad \mu, \nu = 2, \dots, k$ cioè dove i $k-1$ elementi hanno tutti lo stesso label.

Di queste $Q^{(k-1)}$ dicotomie, $R^{(k-1)}(n, p)$ sono quelle identificate da un iperpiano passante anche per il punto escluso ξ_{p+1}^1 .

Delle rimanenti $Q^{(k-1)} - R^{(k-1)}$ dicotomie, solo una frazione di esse ($\tilde{\Psi}_k$) assegna il corretto label a ξ_{p+1}^1 , ossia lo stesso di tutti gli altri $(k-1)$ punti del multipletto $\overline{\xi_{p+1}}$.

Analogamente al caso di input a doppietti:

$$C^{(k)}(n, p+1) = \tilde{\Psi}_k \cdot [Q^{(k-1)}(\dots) - R^{(k-1)}(n, p)] + R^{(k-1)}(n, p) \quad (6.2.6)$$

$\tilde{\Psi}_k$ è una probabilità condizionata e rappresenta tra tutti gli iperpiani che assegnano lo stesso label a tutti i punti del multipletto $\overline{\xi_{p+1}}$, quelli che assegnano lo stesso label a ξ_{p+1}^1 . Un ragionamento più approfondito su questa grandezza è presente in [1].

Le $R^{(k-1)}(n, p)$ dicotomie sono caratterizzate dall'essere identificate da un iperpiano passante per ξ_{p+1}^1 , corrispondono cioè ad un vincolo della dimensione n -esima dello spazio ambiente. Per calcolare tale numero si applica Cover ai restanti $(k-1)$ elementi di $\overline{\xi_{p+1}}$ su $(n-1)$ dimensioni:

$$R^{(k-1)} = Q^{(k-1)}(C^{(k)}(\mathbf{n-1}, p), C^{(k)}(\mathbf{n-2}, p), \dots, C^{(k)}(\mathbf{n-k}, p)) \quad (6.2.7)$$

Unendo infine quest'ultima relazione con (6.2.6) si ottiene l'equazione di ricorrenza per dati strutturati in k -upletti:

$$C^{(k)}(n, p+1) = Q^{(k)}(C^{(k)}(n, p), C^{(k)}(n-1, p), \dots, C^{(k)}(n-k, p)) \quad (6.2.8)$$

dove le $Q^{(k)}$ hanno $k+1$ argomenti e soddisfano la relazione:

$$Q^{(k)}(x_n, \dots, x_{n-k}) = \tilde{\Psi}_k \cdot Q^{(k-1)}(x_n, \dots, x_{n-k}) + (1 - \tilde{\Psi}_k) \cdot Q^{(k-1)}(x_{n-1}, \dots, x_{n-k}) \quad (6.2.9)$$

con $Q^1(x_n, x_{n-1}) = x_n + x_{n-1}$.

Risolvendo l'equazione di ricorrenza in k , si arriva a:

$$C^{(k)}(n, p) = \sum_{j=0}^k \theta_k(j) \cdot C^{(k)}(n-j, p-1) \quad (6.2.10)$$

con $\theta_k(j) = \tilde{\Psi}_k \cdot \theta_{k-1}(j) + (1 - \tilde{\Psi}_k) \cdot \theta_{k-1}(j-1)$

e con condizioni: $\theta_1(0) = \theta_1(1) = 1$ e $\theta_k(-1) = \theta_k(k+1) = 0$

La relazione (6.2.10) rappresenta una formula per il calcolo del numero di dicotomie linearmente separabili su p input strutturati a multipletti di k elementi.

Per quanto riguarda la capacità α_c per i dati strutturati in k -upletti, estendendo il caso di input a doppietti e utilizzando alcune delle considerazioni sui calcoli appena conclusi, nell'articolo [1], si arriva ad una formula chiusa per questa grandezza:

$$\alpha_c = \left(k - \frac{1}{2} - \sum_{j=2}^k \tilde{\Psi}_j \right)^{-1} \quad (6.2.11)$$

6.3 Esempio di applicazione del metodo del simplex

Si è di fronte ad un problema lineare così formulato:

$$\text{Minimizza} \quad K = -2x - 3y - 4z \quad (6.3.1)$$

$$\text{soggetto ai vincoli} \quad 3x + 2y + z \leq 10 \quad (6.3.2)$$

$$2x + 5y + 3z \leq 15 \quad (6.3.3)$$

$$e \quad x, y, z \geq 0 \quad (6.3.4)$$

6.3. ESEMPIO DI APPLICAZIONE DEL METODO DEL SIMPLEX

Si scrive il problema in forma standard, introducendo le variabili di slack e di surplus, per ottenere dei vincoli rappresentati da uguaglianze:

$$3x + 2y + z + s = 10 \quad (6.3.5)$$

$$2x + 5y + 3z + t = 15 \quad (6.3.6)$$

$$x, y, z, s, t \geq 0 \quad (6.3.7)$$

Si sono quindi ottenuti:

$$\vec{c} = \begin{pmatrix} -2 \\ -3 \\ -4 \\ 0 \\ 0 \end{pmatrix}, A = \begin{pmatrix} 3 & 2 & 1 & 1 & 0 \\ 2 & 5 & 3 & 0 & 1 \end{pmatrix} \text{ e } \vec{b} = \begin{pmatrix} 10 \\ 15 \end{pmatrix}.$$

Scrivendo in forma tabellare si ha quindi:

$$\left(\begin{array}{cccccc} & x & y & z & s & t & K \\ \hline 1 & 2 & 3 & 4 & 0 & 0 & 0 \\ 0 & 3 & 2 & 1 & 1 & 0 & 10 \\ 0 & 2 & 5 & 3 & 0 & 1 & 15 \end{array} \right) \quad (6.3.8)$$

Si procede effettuando tutti i passaggi dell'algoritmo:

1. **Verifica di ottimalità** si sta minimizzando: $-\vec{c}^T$ ha tutte le entrate non positive ≤ 0 ? No, le entrate sono tutte positive: il problema non è ottimo.
2. Si sceglie la colonna h , con entrata massima di $-\vec{c}^T$.

Da (6.3.8) si nota subito essere la colonna relativa alla variabile z , ovvero $h = 4$

3. **Verifica di illimitatezza** le entrate della matrice A relative alla colonna 4 sono tutte < 0 ?

Ovvero $a_{i,4} < 0 \quad \forall i = 1, 2$? No:

$$a_{1,4} = 1 > 0 \quad (6.3.9)$$

$$a_{2,4} = 3 > 0 \quad (6.3.10)$$

Il problema risulta quindi limitato.

6.3. ESEMPIO DI APPLICAZIONE DEL METODO DEL SIMPLEX

4. Cerco il *pivot* nella colonna 4:

$$i' = \min\left\{\frac{b_i}{a_{i,4}} : a_{i,4} > 0, \quad \forall i = 1, 2\right\} \quad (6.3.11)$$

$$= \min\left\{\frac{10}{1}, \frac{15}{3}\right\} = \min\{10, 5\} = 5 \quad (6.3.12)$$

Corrispondente a $i' = 2 \rightarrow [a_{2,4} = 3]$ è il termine di pivot

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 0 & 0 & 0 \\ 0 & 3 & 2 & 1 & 1 & 0 & 10 \\ 0 & 2 & 5 & 3 & 0 & 1 & 15 \end{pmatrix} \quad (6.3.13)$$

5. Si effettua l'operazione di cardine, ovvero si sommano o sottraggono multipli interi della riga contenente il *pivot* alle altre righe R che non lo contengono in modo da annullare le entrate della colonna $h = 4$.

La matrice

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 0 & 0 & 0 \\ 0 & 3 & 2 & 1 & 1 & 0 & 10 \\ 0 & 2 & 5 & 3 & 0 & 1 & 15 \end{pmatrix} \quad (6.3.14)$$

in seguito alle operazioni:

$$R_0 - \frac{4}{3}R_2 = 3R_0 - 4R_2 \quad (6.3.15)$$

$$R_1 - \frac{1}{3}R_2 = 3R_1 - R_2 \quad (6.3.16)$$

Diventa:

$$\begin{pmatrix} 3 & -2 & -11 & 0 & 0 & -4 & -60 \\ 0 & 7 & 1 & 0 & 3 & -1 & 15 \\ 0 & 2 & 5 & 3 & 0 & 1 & 15 \end{pmatrix} \quad (6.3.17)$$

Si itera l'algoritmo dall'inizio:

1. **Verifica di ottimalità** si sta minimizzando: $-\bar{c}^T$ ha tutte le entrate non positive ≤ 0 ? Sì, in (6.3.17) si è quindi in presenza della soluzione ottimale per la funzione obiettivo K. Mi concentro sulla prima riga di questa matrice:

$$\begin{pmatrix} x & y & z & s & t & K \\ 3 & -2 & -11 & 0 & 0 & -4 & -60 \end{pmatrix} \quad (6.3.18)$$

Scrivo in forma standard ottenendo:

$$\left(\begin{array}{cccccc|c} & x & y & z & s & t & K \\ 1 & -\frac{2}{3} & -\frac{11}{3} & 0 & 0 & -\frac{4}{3} & \boxed{-20} \end{array} \right) \quad (6.3.19)$$

Il valore riquadrato rappresenta la soluzione estremale della funzione obiettivo, ossia:

$$\boxed{\text{Minimo di } K = -20} \quad (6.3.20)$$

6.4 Esempio di applicazione dell'algoritmo Branch and Bound

Si è di fronte ad una delle casistiche di problemi risolti mediante tecniche di Linear Programming, i cosiddetti problemi *dello zaino*: si ha uno zaino con una data capacità a livello di *peso* e N oggetti caratterizzati ognuno da un peso e da un costo/valore. Si vogliono scegliere gli oggetti da mettere nello zaino in modo da ottimizzarne il valore senza eccedere il peso sostenibile dallo zaino stesso.

$$\begin{cases} \text{MAX } Z = 6x_1 + 3x_2 + 4x_3 + 2x_4 + x_5 \\ 2x_1 + x_2 + 2x_3 + x_4 + x_5 \leq 4 \\ x \in \{0; 1\} \end{cases}$$

Poichè ogni variabile ha un costo c_i e un peso p_i si ordinano le variabili secondo il criterio:

$$\frac{c_1}{a_1} \geq \frac{c_2}{a_2} \geq \dots \geq \frac{c_n}{a_n} \quad (6.4.1)$$

in questo caso specifico, tale criterio è già rispettato, si ha infatti:

$$\frac{6}{2} \geq \frac{3}{1} \geq \frac{4}{2} \geq \frac{2}{1} \geq \frac{1}{1} \quad (6.4.2)$$

Si procede dunque alla determinazione di una soluzione ottima *intera* a cui corrisponde un valore ottimo della funzione obiettivo, una possibile soluzione è:

6.4. ESEMPIO DI APPLICAZIONE DELL'ALGORITMO BRANCH AND BOUND

$$x^* = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \text{ a cui corrisponde un valore della } f \text{ obiettivo pari a } Z^* = 6$$

Sotto queste ipotesi il vincolo è rispettato ma non ottimizzato, valutandolo infatti in x^* si ha:

$$2x_1 + x_2 + 2x_3 + x_4 + x_5 \leq 4 \quad (6.4.3)$$

$$2 \leq 4 \quad (6.4.4)$$

Cerco una soluzione migliore.

$$\text{Per saturare il vincolo pongo } x^* = \begin{pmatrix} 1 \\ 1 \\ \frac{1}{2} \\ 2 \\ 0 \\ 0 \end{pmatrix} \text{ con } Z^* = 11.$$

Tale soluzione è migliore della precedente, ma non è intera.

Genero due sottoproblemi in corrispondenza di x_3 .

Il primo sottoproblema è $P^1 : \bar{x}_3 = \lfloor x_3 \rfloor = \lfloor \frac{1}{2} \rfloor \Rightarrow \bar{x}_3 = 0$

con x_3 nullo si satura il vincolo con:

$$x^{(1)} = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \text{ con } Z^{(1)} = Z(P^1) = 11$$

migliore del risultato precedente in quanto la situazione è intera.

x^{best} diventa quindi $x^{(1)}$ e z^{best} diventa quindi $z^{(1)}$

Il secondo sottoproblema è $P^2 : \bar{x}_3 = \lceil x_3 \rceil = \lceil \frac{1}{2} \rceil \Rightarrow \bar{x}_3 = 1$

con $x_3 = 1$ si satura il vincolo con:

$$x^{(2)} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \text{ con } Z^{(2)} = Z(P^2) = 10$$

$$Z^{(2)} < Z^{best} \quad (6.4.5)$$

$$10 < 11 \quad (6.4.6)$$

$Z^{(2)}$ è peggio del risultato precedente: la scarto e non aggiorno nessun parametro.

La soluzione ottima al problema iniziale è perciò:

$$x = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} \quad \text{con } Z = 11 \quad (6.4.7)$$

Bibliografia

- [1] Rotondo P, Lagomarsino MC, Gherardi M. *Counting the learnable functions of geometrically structured data.* Physical Review Research. 2020 May;2(2). Available from: <http://dx.doi.org/10.1103/PhysRevResearch.2.023169>.
- [2] Gardner E. *Maximum Storage Capacity in Neural Networks.* Europhysics Letters (EPL). 1987 aug;4(4). Available from: <https://doi.org/10.1209/0295-5075/4/4/016>.
- [3] Gardner E, Derrida B. *Optimal storage properties of neural network models.* Journal of Physics A: Mathematical and General. 1988 jan;21(1). Available from: <https://doi.org/10.1088/0305-4470/21/1/031>.
- [4] Cover TM. *Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition.* IEEE Trans on Electronic Computers. 1965;EC-14.
- [5] Perron L, Furnon V. *OR-Tools;*. Available from: <https://developers.google.com/optimization/>.
- [6] Borra F, Cosentino Lagomarsino M, Rotondo P, Gherardi M. *Generalization from correlated sets of patterns in the perceptron.* Journal of Physics A: Mathematical and Theoretical. 2019 Aug;52(38):384004. Available from: <http://dx.doi.org/10.1088/1751-8121/ab3709>.
- [7] Mehta P, Bukov M, Wang CH, Day AGR, Richardson C, Fisher CK, et al. *A high-bias, low-variance introduction to Machine Learning for physicists.* Physics Reports. 2019 May;810. Available from: <http://dx.doi.org/10.1016/j.physrep.2019.03.001>.
- [8] Chung S, Lee DD, Sompolinsky H. *Classification and Geometry of General Perceptual Manifolds.* Physical Review X. 2018 Jul;8(3). Available from: <http://dx.doi.org/10.1103/PhysRevX.8.031003>.

- [9] Chung S, Lee DD, Sompolinsky H. *Linear readout of object manifolds*. Physical Review E. 2016 Jun;93(6). Available from: <http://dx.doi.org/10.1103/PhysRevE.93.060301>.
- [10] Engel A, Van den Broeck C, Broeck C. *Statistical Mechanics of Learning*. Statistical Mechanics of Learning. Cambridge University Press; 2001. Available from: <https://books.google.it/books?id=qVo4IT9ByfQC>.
- [11] Feldman J, Rojas R. *Neural Networks: A Systematic Introduction*. Springer Berlin Heidelberg; 2013. Available from: <https://books.google.it/books?id=4rESBwAAQBAJ>.
- [12] Geszti T. *Physical Models of Neural Networks*. World Scientific; 1990. Available from: <https://books.google.it/books?id=PHJDmYhvTMsc>.
- [13] Hu B, Lu Z, Li H, Chen Q. *Convolutional Neural Network Architectures for Matching Natural Language Sentences*; 2015.
- [14] Rajesh PN Rao BAOeMSL. *Probabilistic Models of the Brain: Perception and Neural Function*. MIT Press; 2002.
- [15] Cohen CSLDDSH Uri. *Separability and geometry of object manifolds in deep neural networks*. Nature Communications. 2020 feb;11. Available from: <https://doi.org/10.1038/s41467-020-14578-5>.
- [16] Gherardi M. *Solvable Model for the Linear Separability of Structured Data*. Entropy. 2021;23(3). Available from: <https://www.mdpi.com/1099-4300/23/3/305>.
- [17] LeCun BYHG Yann. *Deep learning*. Nature. 2015 may;521. Available from: <https://doi.org/10.1038/nature14539>.
- [18] Si J, Barto AG, Powell WB, Wunsch D. In: *The Linear Programming Approach to Approximate Dynamic Programming*; 2004. p. 153–178.
- [19] Vanderbei RJ. *Linear Programming*. Springer; 2014.
- [20] Zdeborová L. *Understanding deep learning is also a job for physicists*. Nature Physics. 2020 may;16(6):602–604.

BIBLIOGRAFIA

- [21] Lopez B, Schroder M, Opper M. *Storage of correlated patterns in a perceptron.* Journal of Physics A: Mathematical and General. 1995 Aug;28(16):L447–L452. Available from: <http://dx.doi.org/10.1088/0305-4470/28/16/005>.