Dipartimento di Impresa e Management

Corso di Laurea Magistrale in Marketing

Cattedra di Machine Learning & object driven Marketing

Reti neurali e collaborative filtering: un approccio avanzato al marketing one-to-one

Relatore:

Prof. Luigi Laura

Candidato:

Pasquale De Rosa

Matr. 686081

Correlatore:

Prof. Giuseppe Francesco Italiano

Anno Accademico 2017/18

A mio nonno Pasquale,
fonte inesauribile di idee
e di saggezza

Indice

1 Introduzione 4

2 Sistemi di raccomandazione e marketing one-to-one 6

2.1 Cosa sono i sistemi di raccomandazione? 7

2.2 Tipologie di sistemi di raccomandazione 9

2.2.1 Collaborative filtering 9

2.2.2 Content-based filtering 15

2.2.3 Sistemi ibridi di raccomandazione 19

2.3 L'importanza dei sistemi di raccomandazione per il marketing one-to-one 20

3 Le reti neurali artificiali: tipologie e generalità 23

3.1 Dalla rete neurale biologica alla rete neurale artificiale 25

3.2 Una prima forma di rete neurale artificiale: il percettrone 28

3.3 Dallo "shallow learning" al "deep learning": il percettrone multistrato . 33

3.4 Le reti neurali convoluzionali 38

3.5 Modelli "feed-back" e reti neurali ricorrenti 44

4 Metodologie per l'addestramento di reti neurali 48

4.1 Funzioni di costo e scelta dei parametri del modello 49

4.2 Algoritmi di minimizzazione della funzione di costo: discesa del gradiente e SGD 53

4.3 Un algoritmo per la computazione del gradiente: la backpropagation . . 59

4.4 Addestrare reti neurali "feed-back" attraverso la backpropagation through time 63

5 Collaborative filtering: un confronto empirico tra algoritmi "standard" e reti neurali 69

5.1 Il dataset di riferimento: Movielens 100K 70

5.2 L'approccio classico al collaborative filtering: k-Nearest Neighbors (k-NN) e Decomposizione ai valori singolari (SVD) 73

5.2.1 Cos'è il k-NN? 73

5.2.2 k-NN e collaborative filtering: descrizione della ricerca e analisi dei risultati 76

5.2.3 Fattorizzazione della matrice e SVD 78

5.2.4 SVD e collaborative filtering: decrizione della ricerca e analisi dei risultati 81

5.3 L'approccio "neurale" al collaborative filtering: Reti "feed-forward" e LSTM 83

5.3.1 Reti "feed-forward" e collaborative filtering: descrizione della ricerca e analisi dei risultati 83

5.3.2 LSTM e collaborative filtering: descrizione della ricerca e analisi dei risultati 88

5.4 Approcci a confronto e esiti della ricerca 90

6 Conclusioni 93

A Lista dettagliata dei programmi e degli algoritmi 95

A.1 k-NN 95

A.2 SVD 97

A.3 Rete neurale "feed-forward" 98

A.4 LSTM 102

Bibliografia 104

Capitolo 1

Introduzione

Il presente elaborato mira a discutere l'utilizzo di un paradigma innovativo, basato sulle c.d. "reti neurali artificiali", ai fini della costruzione di sistemi di raccomandazione avanzati, attraverso un confronto tra questa complessa "famiglia" di strumenti algoritmici ed approcci maggiormente "tradizionali".

Naturalmente, essendo l'oggetto della trattazione in esame particolarmente complesso ed articolato, i primi capitoli di questa tesi saranno incentrati sui fondamenti teorici dei sistemi di raccomandazione (cap. 2) e delle reti neurali artificiali (capp. 3-4), fornendo un'esaustiva panoramica sui concetti alla base di ciascuno di questi argomenti, e offrendo al lettore un necessario "background" per la comprensione della successiva analisi del mio lavoro di ricerca individuale in materia.

Le ragioni che mi hanno condotto alla scelta di tale argomento di studio risiedono nel mio personale interesse per le reti neurali, rappresentanti una delle frontiere maggiormente rivoluzionarie nel campo dell'intelligenza artificiale, e dalla volontà, in ragione del mio personale curriculum studiorum, di analizzare un possibile utilizzo delle stesse ai fini di creare avanzati strumenti di marketing personalizzato. A tal proposito, il quinto capitolo di questa tesi analizzerà nel dettaglio l'efficacia del connubio tra i due sovracitati ambiti di ricerca, attraverso la descrizione di un'applicazione pratica di tali

modelli per la costruzione di un'architettura volta a fornire ad un generico consumatore consigli sui propri possibili acquisti futuri, nota in letteratura anche con il nome di "sistema di raccomandazione".
In ragione di ciò, le performance ottenute utilizzando dei modelli di rete neurale saranno comparate con quelle generate dall'applicazione di altre tecniche, maggiormente consolidate per la risoluzione di questa specifica tipologia di problemi, ma dotate di minore complessità e sofisticatezza.

Capitolo 2

Sistemi di raccomandazione e marketing one-to-one

Nel 2006, il colosso californiano dell'intrattenimento Netflix lanciò una competizione che di lì a poco sarebbe passata alla storia: un milione di dollari a chi avrebbe migliorato del 10% le performance del sistema di raccomandazione utilizzato dalla compagnia, CineMatch. La competizione durò ben tre anni, finchè nel settembre 2009 i "BellKor's Pragmatic Chaos" si aggiudicarono l'agognato premio creando un algoritmo con un errore quadratico medio (RMSE) pari a 0.8567, inferiore a quello prodotto da CineMatch del 10.06% [36].

Da allora, abbiamo assistito in tutto il mondo a un vero e proprio boom dei sistemi di raccomandazione, ormai ampiamente utilizzati come principale strumento di marketing one-to-one dalle più importanti aziende del pianeta. Basti pensare che nel 2013 uno studio condotto dalla nota società di consulenza manageriale McKinsey stimò che il 35% delle transazioni effettuate su Amazon, così come il 75% dei film guardati su Netflix, fosse attribuibile ai sistemi di raccomandazione [20].

Ma in cosa consisteva esattamente il funzionamento di CineMatch? E, più in generale, cos'è un sistema di raccomandazione? Quali sono le ragioni che possono condurre una multinazionale come Netflix ad investire ben un milione di dollari per migliorare le

proprie performance in questo campo? Per quale ragione l'utilizzo di tali algoritmi può costituire una fonte rilevante di vantaggio competitivo per le aziende?
Il presente capitolo mira a dare una risposta esaustiva a ciascuna di queste domande, partendo da una definizione chiara e rigorosa di sistema di raccomandazione per poi descriverne le tipologie e le principali applicazioni al marketing e al contesto aziendale.

2.1 Cosa sono i sistemi di raccomandazione?

Nel marketing tradizionale si suole schematizzare il complesso sistema di interazioni tra cliente e impresa in un modello gerarchico, che da un'esposizione iniziale ad uno stimolo commerciale procede verso il formarsi di determinati atteggiamenti verso un prodotto, fino a giungere al suo possibile acquisto [25, pp. 45-46]. In tal senso, il modo e il tempo in cui il consumatore viene a conoscenza dell'esistenza di un particolare bene, che possa soddisfare un suo bisogno inespresso, gioca un ruolo fondamentale.
Nell'era dei "big data" e della personalizzazione di massa, le aziende tendono ad affinare sempre più gli strumenti a loro disposizione per far sì che i propri potenziali clienti possano imbattersi esattamente in ciò che desiderano, nel preciso momento in cui insorge il bisogno; è in tale contesto che nascono e si sviluppano i sistemi di raccomandazione.
Un sistema di raccomandazione è, essenzialmente, uno strumento che permetta di semplificare il processo di decision making di un consumatore, fornendogli suggerimenti su cosa acquistare. Il fine di questo genere di strumenti è evidente: attraverso una politica commerciale più mirata, le aziende possono incrementare significativamente il volume delle proprie vendite, cercando le migliori corrispondenze tra le caratteristiche dei prodotti in portafoglio con i desideri dei propri clienti attuali e potenziali.
Tali strumenti poggiano su un complesso sistema di dati e informazioni riguardanti i clienti, i prodotti da raccomandare e le transazioni, ossia le interazioni passate tra clienti e prodotti; il volume e la tipologia delle informazioni raccolte varia a seconda della complessità del sistema di raccomandazione che ci si propone di costruire [15, pp. 7-8]. Sugli utenti, ad esempio, è possibile raccogliere informazioni complesse e dettagliate quali variabili di natura demografica, psicografica o sociale; i prodotti possono

essere descritti da variabili che ne evidenzino la tipologia e le caratteristiche principali, e le stesse transazioni possono essere registrate nella forma di dati maggiormente strutturati, come giudizi sintetici espressi in forma di rating, o dati non strutturati come il contenuto delle recensioni [15, pp. 8-10].

Ma in che modo le imprese utilizzano tali dati per creare delle efficaci raccomandazioni? Un primo approccio può essere quello di analizzare le diverse transazioni per individuare gruppi di clienti con preferenze simili tra loro; in tal modo, se il cliente X e il cliente Y hanno manifestato dei comportamenti d'acquisto analoghi nel passato, si potranno raccomandare a ciascuno di loro i prodotti già graditi dall'altro.

Tale approccio viene anche definito "Collaborative filtering" [30, p. 3], e sarà più chiaro una volta osservato l'esempio seguente: si pensi ad una generica azienda di nome XYZ, operante nel largo consumo, che desideri implementare un sistema di raccomandazione per migliorare le performance del proprio canale e-commerce. Avendo a disposizione un ampio database contenente lo storico delle transazioni effettuate negli ultimi dieci anni, decide di organizzarlo in modo tale da mettere in luce eventuali strutture di gruppo nei propri clienti, che saranno poi utilizzate per suggerire a ciascun cliente dei possibili futuri acquisti.

In questo caso l'obiettivo di XYZ è costruire un sistema di raccomandazione che tenga in considerazione dei pattern di similarità esistenti nel proprio portafoglio di clienti: è dunque un classico esempio di Collaborative filtering.

Ma supponiamo invece che la sopracitata azienda decida di utilizzare unicamente l'insieme di dati riferiti a ciascun cliente per costruire le proprie raccomandazioni, senza seguire un approccio "collaborativo" tra i consumatori presenti nel proprio portafoglio. In questo caso XYZ sarà interessata ad evidenziare dei pattern di similarità tra i prodotti acquistati da ciascun cliente negli ultimi dieci anni, in modo tale da comprendere la struttura delle sue preferenze ed effettuare efficaci raccomandazioni per i suoi acquisti futuri.

Questo tipo di approccio alla costruzione dei sistemi di raccomandazione è anche detto Content-based filtering [30, p. 3], in quanto incentrato sulle analisi del contenuto delle

singole transazioni effettuate dall'azienda con i propri clienti.
Collaborative filtering e Content-based filtering non esauriscono, tuttavia, la gamma di strumenti a disposizione delle aziende per effettuare delle raccomandazioni mirate ai propri clienti. Come si vedrà più avanti, l'azienda XYZ potrebbe optare per uno strumento ibrido tra i due, che mantenga i vantaggi offerti da entrambi cercando di eliminarne, al contempo, i difetti; tali sistemi di raccomandazione vengono definiti per l'appunto ibridi [30, p. 3], in quanto combinano in un unico approccio le tecniche utilizzate per il Collaborative filtering e per il Content-based filtering.
Nella prossima sezione descriverò dettagliatamente ciascuna di queste tre tipologie di sistemi di raccomandazione, fornendo al contempo un'ampia panoramica sui principali metodi algoritmici utilizzati per implementarli.

2.2 Tipologie di sistemi di raccomandazione

Questa sezione, come anticipato in precedenza, descriverà in maniera maggiormente dettagliata le tre tipologie più comuni di sistemi di raccomandazione e fornirà una visione d'insieme dei principali strumenti algoritmici utilizzabili per la loro costruzione. Nello specifico, la sezione 2.2.2 decriverà l'approccio noto come Collaborative filtering, già precedentemente accennato, il quale si basa sul concetto di similarità tra utenti e che sarà successivamente utilizzato per il mio lavoro di ricerca empirica; le sezioni 2.2.3 e 2.2.4 descriveranno invece gli approcci Content-based e ibridi, anch'essi molto utilizzati nella pratica comune.

2.2.1 Collaborative filtering

Per "Collaborative filtering" s'intende un'approccio alla raccomandazione basato sulla convergenza tra le preferenze di utenti diversi: a ciascuno di essi saranno raccomandati prodotti che abbiano già incontrato il gradimento di utenti con gusti simili [15, p. 11] [30, p. 4]. Un esempio molto chiaro del funzionamento di questa tipologia di sistemi di raccomandazione è dato dalla figura in basso: se i due utenti hanno caratteristiche

simili, e ciascuno di essi ha acquistato le stesse pietanze, potremo raccomandare al secondo cliente la stessa bevanda acquistata dal primo, in ragione della simmetria tra le rispettive preferenze.

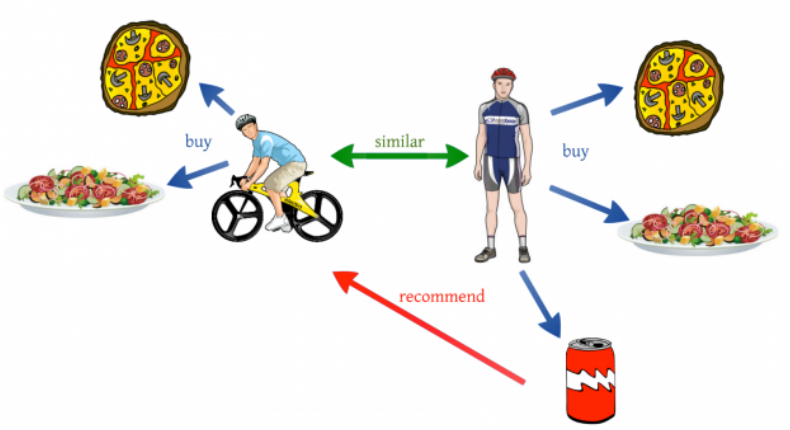

Figura 2.1. Esempio di Collaborative filtering [31].

Ma quali sono le ragioni che possono condurre un'azienda a preferire un approccio di questo genere alla raccomandazione, piuttosto che un sistema Content-based oppure ibrido?

Innanzitutto un sistema di Collaborative filtering può "estendere" l'orizzonte d'acquisto di ciascun utente a tipologie prodotti che egli potrebbe non aver mai considerato, in quanto le raccomandazioni sono basate su un confronto collaborativo tra clienti e non su un'analisi delle preferenze gà espresse dal singolo cliente [39, p. 88]. Questo significa che, attraverso raccomandazioni basate su preferenze altrui, ciascun utente potrebbe "scoprire" nuove tipologie di prodotto, acquisendo nuovi interessi e permettendo all'azienda di differenziare e incrementare la proposta originaria di valore nei suoi riguardi.

Un ulteriore vantaggio che i sistemi di Collaborative filtering presentano in confronto ad altre forme di raccomandazione è la loro flessibilità, unita alla comprovata efficacia per tipologie più "complesse" di prodotti quali quelli musicali o cinematografici [39, p. 88].

La presenza di questi vantaggi rende sicuramente i sistemi di raccomandazione "collaborativi" una delle forme più efficienti ed utilizzate da parte delle aziende, anche in

presenza di taluni difetti; tra questi, vale senza dubbio la pena citare:

- Il problema della "partenza a freddo" (cold-start), che si riscontra per i prodotti nuovi, i quali generalmente presentano uno scarso numero di rating da parte degli utenti, rendendo impossibile la creazione di sistemi di raccomandazione di natura "collaborativa" [39, pp. 88-89];

- La sparsità delle informazioni relative ai rating, che per ciascun utente saranno costituiti per lo più da valori mancanti, in quanto è oggettivamente impossibile che uno stesso cliente possa attribuire un giudizio a ciascun prodotto offerto dall'azienda [39, pp. 88-89];

- La necessità di un enorme volume di dati relativi a ciascun utente per costruire degli efficaci sistemi di raccomandazione di questa tipologia [39, pp. 88-89].

Seppure il funzionamento di questi strumenti possa apparire a prima vista banale, in realtà per permetterne l'implementazione pratica è richiesto l'utilizzo di tecniche sufficientemente complesse e avanzate.
Uno strumento di analisi largamente impiegato per tale proposito è la matrice "user/-rating" nella quale le preferenze dei clienti di un'azienda vengono rappresentate da una matrice $m \times n$, dove m è il numero degli utenti e n il numero dei prodotti (o items) verso cui ciascun utente ha espresso un giudizio sintetico, nella forma di un rating [30, p. 3].

$$R = \begin{pmatrix} r_{11} & r_{12} & r_{13} & \dots & r_{1n} \\ r_{21} & r_{22} & r_{23} & \dots & r_{2n} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ r_{m1} & r_{m2} & r_{m3} & \dots & r_{mn} \end{pmatrix}$$

Come si può facilmente osservare, ciascun elemento della matrice è costituito da un rating, espresso da un particolare utente su un determinato prodotto d'interesse (ad

esempio, r_{11} è il rating espresso dal primo cliente sul primo prodotto, così come sono stati classificati all'interno del database aziendale).
La matrice user/rating è, per sua natura, una matrice sparsa, ossia costituita per lo più da zeri e valori mancanti, in quanto è estremamente difficile che ciascun utente abbia potuto esprimere un rating per ogni singolo prodotto venduto dall'azienda [30, p. 3].
Utilizzando la matrice user/rating come punto di partenza, è possibile costruire degli algoritmi che permettano, partendo da una struttura nota delle preferenze dei vari utenti, di effettuare predizioni sui rating che essi attribuirebbero a prodotti sui quali non hanno ancora espresso un gradimento. Tali predizioni potranno poi, come è chiaro, essere utilizzate per definire la base delle successive raccomandazioni.
Gli algoritmi di Collaborative filtering possono essere ricondotti a due famiglie: tecniche Neighborhood-based e tecniche Model-based [30, p. 4]; nella restante parte di questa sezione cercherò di fornire una panoramica sufficientemente esaustiva di entrambe, rimandando tuttavia ai capitoli successivi per una trattazione più approfondita delle metodologie che in questa sede verranno unicamente accennate (k-Nearest Neighbors, SVD, reti neurali).

Neighborhood-based Collaborative filtering

I metodi "Neighborhood-based" si basano sul concetto di "nearest neighbors" per effettuare delle predizioni sui rating che un dato utente esprimerà su un insieme di prodotti [30, p. 4]. Tali "neighbors" non sono altro che un sottoinsieme costituito dai k utenti più simili al consumatore in questione, i cui rating saranno poi calcolati come combinazione "pesata" dei rating già espressi per gli stessi prodotti dai "nearest neighbors" [30, p. 4].
Questa famiglia di metodi trae il proprio nome dall'algoritmo di apprendimento supervisionato k-Nearest Neighbors[1], per il quale, una volta definita una generica misura di

[1]Per una più chiara e rigorosa spiegazione dell'algoritmo e delle sue applicazioni al Collaborative filtering, si consultino le sezioni 5.2.1 e 5.2.2.

similarità ω_{au}[2] tra un utente a e ciascun utente $u \neq a$, il rating predetto per l'utente a relativo al prodotto p sarà uguale a [15, p. 115]:

$$\hat{r_{ap}} = \frac{\sum_{u \in k_p} \omega_{au} \cdot r_{up}}{\sum_{u \in k_p} \omega_{au}} \tag{2.1}$$

Dove k_p è il numero di utenti u che sono "nearest neighbors" di a e che hanno attribuito un rating a p, r_{up} il rating reale attribuito da ciascun utente u al prodotto p, e ω_{au}, come già anticipato in precedenza, una misura di similarità.
L'espressione $\sum_{u \in k_p} \omega_{au}$ compare anche a denominatore per garantire una nomalizzazione dei pesi, che altrimenti potrebbero uscire dal range dei valori consentiti [15, p. 115].
In un'ottica più propriamente di marketing, l'utilizzo di metodi basati sul concetto di "neighborhood" per il Collaborative filtering è legata al principio del "word-of-mouth" [15, p. 114], per cui buona parte degli individui, al momento della valutazione di un possibile acquisto, si affida al parere di altri individui, ritenuti affidabili e aventi gusti assimilabili ai propri.
A dispetto della loro straordinaria semplicità concettuale, le tecniche di raccomandazione "Neighborhood-based" risultano tuttavia largamente utilizzate per costruire sistemi di Collaborative filtering [15, p. 139]. Ciò è dovuto agli innumerevoli vantaggi che presentano tali metodi, tra i quali si possono citare:

- Efficienza computazionale, che rende l'uso di tali tecniche appropriato quando si ha a che fare con database molto vasti, ai quali possono adattarsi in maniera molto semplice e rapida [15, pp. 119-120];

- Comprovata stabilità di tali metodi rispetto a variazioni (anche particolarmente frequenti) nel database [15, pp. 119-120];

[2]Come si vedrà nella sezione 5.2.1, esistono diverse misure di similarità tra utenti, le più utilizzate delle quali sono la similarità del coseno, il coefficiente di Pearson e il mean-squared difference (MSD).

- "Serendipità", intesa come capacità di suscitare nel cliente l'interesse per prodotti nuovi o in quel momento sconosciuti, per il semplice motivo che essi sono stati apprezzati in precedenza dai k "neighbors" più simili a lui [15, pp. 119-120].

Model-based Collaborative filtering

Le metodologie "Model-based" fanno uso di tecniche e modelli statistici per stimare i rating forniti da un particolare utente [30, p. 7]; tra questi possiamo citare gli algoritmi di classificazione e regressione, spesso utilizzati insieme a tecniche di riduzione della dimensionalità (come l'Analisi in componenti principali), volte a pre-processare la matrice user/rating allo scopo di diminuirne la sparsità [30, p. 7].
Tuttavia, gli algoritmi standard di Machine Learning non esauriscono del tutto l'insieme di metodologie statistiche utilizzabili per costruire dei modelli di Collaborative filtering. Un ruolo decisamente importante a tal scopo lo rivestono i c.d. "Modelli dei fattori latenti", tecniche volte a "estrapolare" dal rating esplicito attribuito da un utente i "fattori latenti" che ne sono alla base e che hanno caratterizzato la sua decisione [15, p. 146].
Appartengono a questa famiglia di modelli le tecniche di fattorizzazione (o decomposizione) della matrice, la principale delle quali è senza dubbio la Decomposizione ai valori singolari (SVD)[3], che si basa sull'assunto di fondo che la similarità tra due o più utenti è determinata da strutture nascoste presenti nei dati [30, p. 7]; per tale ragione, sia gli utenti che i prodotti possono essere rappresentati da due vettori $\overline{u}$ e $\overline{p} \in R^k$ (dove k sono le dimensioni "latenti"), e i rating stimati saranno calcolati attraverso il prodotto scalare $\overline{u}^T \cdot \overline{p}$ [30, p. 7]. In tal modo, si riprodurrà esattamente una matrice user/rating della tipologia già vista all'inizio di questa sezione, dove però gli elementi della stessa saranno costituiti dalle predizioni risultanti dal prodotto scalare tra il vettore degli utenti e quello dei prodotti.
Ma in che modo possiamo "vettorizzare" utenti e prodotti in modo da esplicitarne la

[3]Per una trattazione maggiormente approfondita dell'algoritmo si rimanda alle sezioni 5.2.3 e 5.2.4.

struttura latente? Come vedremo in seguito, nella sezione 5.2.3, nel 2006, un partecipante al già citato "premio Netflix" (noto con il nickname "Simon Funk"), propose una variante dell'algoritmo standard di SVD [8] accompagnato ad un metodo di ottimizzazione, come la discesa stocastica del gradiente (SGD), che permettesse di creare degli "embeddings"[4] di utenti e prodotti tali da consentire la minimizzazione di una determinata funzione obiettivo di costo[5].

Anche le reti neurali artificiali possono essere utilizzate in modo tale da esplicitare delle strutture latenti nei dati[6] e dunque, di conseguenza, rappresenterebbero un esempio ulteriore di "Modello dei fattori latenti" [15, p. 151].

Anche questa tipologia di sistemi di raccomandazione, così come quella analizzata in precedenza, trova largo utilizzo pratico tra le aziende; ciò è dovuto, innanzitutto, alla grande flessibilità di questi strumenti che, essendo basati su un complesso variegato di modelli statistici, possono trovare applicazione in diversi problemi reali. Inoltre, le tecniche di Collaborative filtering "Model-based" hanno mostrato una notevole capacità predittiva, superiore anche ai sistemi "Neighborhood-based", come dimostrato dalle ottime performance ottenute in competizioni come il "premio Netflix" [15, p. 161].

2.2.2 Content-based filtering

L'approccio alla raccomandazione noto come "Content-based filtering" non prevede l'utilizzo della matrice user/rating come punto di partenza per la costruzione di modelli predittivi, volti all'analisi del comportamento futuro di un cliente basandosi sullo studio delle transazioni svolte in passato con altri utenti simili a lui [30, pp. 9-10]; a differenza di quanto visto per il Collaborative filtering, infatti, le raccomandazioni sono effettuate seguendo un approccio maggiormente "personalizzato", e basato unicamente sugli acquisti precedenti e le caratteristiche psico-socio-demografiche del singolo utente

[4] Vettori che esplicitano delle strutture nascoste nei dati.

[5] Per maggiori informazioni sulle funzioni di costo e l'algoritmo di SGD, si faccia riferimento alle sezioni 4.1 e 4.2.

[6] Attraverso degli "strati di embedding" iniziali, come verrà poi spiegato dettagliatamente nella sezione 5.3.1.

[30, pp. 9-10].

Un sistema di raccomandazione di questa tipologia mira ad analizzare l'insieme dei prodotti sui quali un determinato utente ha espresso in passato un rating, allo scopo di costruire, per ciascun cliente in portafoglio per l'azienda, un "profilo" che ne mappi le preferenze e gli interessi [15, p. 75]; una volta fatto ciò, il compito dell'impresa sarà soltanto quello di comprendere quali prodotti siano maggiormente adatti a soddisfare un determinato profilo di cliente [15, p. 75].

Per comprendere meglio quanto detto in precedenza, si faccia riferimento alla figura in basso: se l'utente in questione ha dato prova in passato di apprezzare una determinata tipologia di bevanda, allora l'azienda potrà raccomandare lui un altro marchio di bevande che presenti caratteristiche simili a quello per cui egli ha già espresso il proprio gradimento.

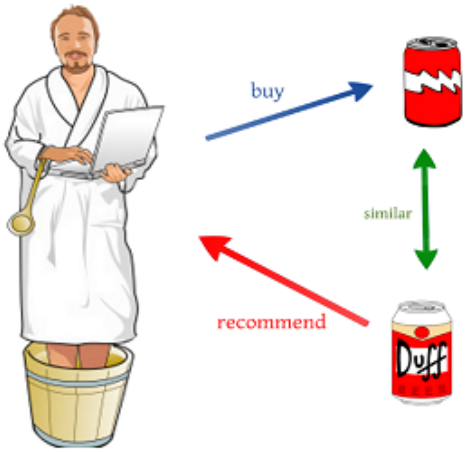

Figura 2.2. Esempio di Content-based filtering [31].

Naturalmente anche l'approccio "Content-based" presenta diversi vantaggi, se comparato alle tecniche di Collaborative filtering. Innanzitutto, permette di effettuare raccomandazioni indipendenti per i singoli utenti, senza necessità di analizzare il comportamento di altri clienti simili ai primi [15, p. 78]; inoltre questa tipologia di sistemi

di raccomandazione non soffre del c.d. problema della "partenza a freddo", ossia della scarsa capacità predittiva per prodotti di nuova immissione sul mercato, in quanto i suggerimenti si basano unicamente sulle preferenze del singolo utente, e sono indipendenti dal giudizio degli altri [15, p. 78].
Tuttavia, anche le tecniche di Content-based filtering presentano alcuni svantaggi strutturali, tra i quali si possono senza dubbio annoverare:

- L'incapacità delle raccomandazioni di suggerire categorie "nuove" di prodotti, che non abbiano punti in comune con quelli per i quali l'utente ha già espresso dei giudizi [15, pp. 78-79];

- L'inefficacia predittiva di tali sistemi laddove vi siano nuovi utenti, con un numero limitato di rating espressi per ciascun prodotto acquistato in passato [15, pp. 78-79].

Ultimamente si è assistito ad un incremento significativo dei sistemi di Content-based filtering basati su recensioni e altre forme di dati non strutturati, come i documenti testuali [30, p. 10], dovuto alla forte diffusione di portali e applicazioni del web volte a permettere la diffusione e la condivisione di tali contenuti da parte degli utenti.
Ma in che modo le aziende possono estrarre da recensioni e altre tipologie di dati testuali le informazioni di cui necessitano per descrivere le preferenze di un utente?
Un approccio molto comune, noto come Vector Space Model (VSM), tende a rappresentare ciascun documento testuale come un vettore nello spazio n-dimensionale [15, p. 81]:

$$d_i = \{w_{1i}, w_{2i}, w_{3i}, ..., w_{ni}\} \tag{2.2}$$

Dove w_{1i} è il peso del primo termine t_{1i} del documento d_i; i "pesi" di ciascun termine presente nella recensione vengono determinati attraverso una tecnica nota come "pesatura TF-IDF"[7], dove si assume che termini che si presentano con frequenza in uno

[7]Dove "TF" sta per "Term Frequency" e "IDF" per "Inverse Document Frequency".

specifico documento/recensione, e con minore probabilità nell'intero corpus[8], potrebbero avere maggiore rilevanza e attinenza con l'argomento della discussione [15, p. 81]. Più specificamente, seguendo l'approccio TF-IDF, ciascun peso w_{ki} può essere calcolato come [15, p. 82]:

$$w_{ki} = \text{TF-IDF}(t_k, d_i) \tag{2.3}$$

Dove [15, p. 82]:

$$\text{TF-IDF}(t_k, d_i) = \frac{f_{k,i}}{max_z f_{z,i}} \cdot log \frac{N}{n_k} \tag{2.4}$$

Il primo termine $\frac{f_{k,i}}{max_z f_{z,i}}$, corrispondente alla "Term Frequency" (TF), indica semplicemente la frequenza relativa del k-esimo termine presente nel documento i rispetto alla massima frequenza $max_z f_{z,i}$ computata per tutti gli z termini presenti nel documento i; il secondo termine $log \frac{N}{n_k}$, corrispondente alla "Inverse Document Frequency", indica invece l'inverso (calcolato su scala logaritmica) del numero di documenti n_k in cui il k-esimo termine t_k compare almeno una volta, rispetto al numero complessivo di documenti nel corpus N [15, p. 82].

In alternativa all'approccio VSM, risultano essere largamente utilizzate anche altre tecniche di classificazione testuale nell'ambito del Content-based filtering: una di queste è il c.d. "Classificatore Bayesiano Naif", o Naïve Bayes, che calcola la probabilità di un documento d di appartenere a una determinata classe c come [15, p. 91]:

$$P(c \mid d) = \frac{P(c) \cdot P(d \mid c)}{P(d)} \tag{2.5}$$

Questo algoritmo, mutuato dal noto Teorema della probabilità condizionata di Bayes[9], è uno dei classificatori più semplici utilizzati nel campo dell'apprendimento supervisionato, ma al contempo conserva una comprovata efficacia nella risoluzione di problemi legati alla catalogazione testuale, come ad esempio i c.d. filtri anti-spam o, nel caso

[8] Insieme di documenti soggetti all'analisi e allo studio delle opinioni.

[9] Noto teorema statistico formulato da Thomas Bayes (1702-1761) e pubblicato postumo nel 1763, per il quale la probabilità di un evento A condizionata da un altro evento B, $P(A \mid B)$, è calcolabile se sono note $P(A)$, $P(B)$ e $P(B \mid A)$, come $\frac{P(A) \cdot P(B|A)}{P(B)}$.

in oggetto, l'estrazione di documenti da un corpus che contengano caratteristiche di interesse per effettuare raccomandazioni "Content-based" [15, p. 91].
Seguendo questo approccio, la categoria $c \in C$ di appartenenza di un documento d può essere determinata calcolando quella cui è attribuita la maggiore probabilità secondo la (2.5), come segue [15, p. 91]:

$$c = argmax_{c \in C} \frac{P(c) \cdot P(d \mid c)}{P(d)} \tag{2.6}$$

L'algoritmo Naïve Bayes assume l'indipendenza statistica tra ciascuna parola contenuta in un dato documento d, data la classe c di appartenenza [15, p. 91]. Tale condizione tuttavia è generalmente in contrasto con quanto avviene nella realtà, rendendo il presupposto di base dell'indipendenza condizionale, su cui l'algoritmo si basa, sostanzialmente ingenuo e "forzato" (da qui l'appellativo di classificatore "naïve"); ciononostante, questo strumento si è dimostrato particolarmente efficace nell'ambito della classificazione dei documenti testuali, riuscendo ad ottenere delle performance superiori anche rispetto ad algoritmi maggiormente complessi e sofisticati [15, p. 91].

2.2.3 Sistemi ibridi di raccomandazione

I sistemi di raccomandazione "collaborativi" e "content-based" presentano, come si è visto in precedenza, un elevato numero di vantaggi ma anche qualche limite; per tale ragione, negli ultimi tempi le ricerche in materia tendono a focalizzarsi maggiormente su come combinare questi due approcci in modo da esaltarne i punti di forza e contenerne quelli di debolezza [30, p. 10]. È in tale contesto che si sono sviluppati i c.d. sistemi di raccomandazione "ibridi", ossia tecniche che uniscono gli strumenti di Collaborative filtering e quelli di Content-based filtering allo scopo di incrementare notevolmente la capacità predittiva rispetto ai risultati ottenibili con entrambi.
Nel presente paragrafo saranno brevemente passati in rassegna soltanto alcuni dei principali contributi in materia, essendo la complessità dell'argomento tale da non permetterne una trattazione più approfondita in questa sede.

Un primo spunto, concettualmente molto semplice, è offerto dal lavoro di Paul Cotter e Barry Smith [28, p. 5] [30, p. 10], i quali hanno proposto un sistema di raccomandazione "misto" per il suggerimento di programmi televisivi, una parte dei quali è selezionata attraverso un modello di Collaborative filtering, ed una seconda parte utilizzando degli strumenti di Content-based filtering.
Un secondo approccio che vale la pena di citare è quello fornito da Prem Melville et al. [29, pp. 3-6] [30, p. 11] i quali propongono un modello di "Content-boosted Collaborative filtering", in cui si rimedia alla sparsità della matrice user/rating utilizzando l'algoritmo Naïve Bayes per rimpiazzare i valori mancanti relativi ai rating espressi dagli utenti, per poi costruire le raccomandazioni vere e proprie attraverso un modello di Collaborative filtering addestrato sulla nuova matrice precedentemente ottenuta.
Un ulteriore contributo è fornito dallo studio di Michael J. Pazzani [24, pp. 9-10] [30, p. 11] il quale propone un modello di "Collaboration via content", in cui vengono utilizzate delle tecniche di Content-based filtering per costruire dei "profili utente" sui quali sono poi applicati direttamente gli algoritmi di Collaborative filtering, senza dunque costruire una matrice user/rating.
Infine, particolarmente interessante risulta essere il lavoro di Basu et al. [5, pp. 2-7] [30, p. 11], i quali propongono un approccio misto in cui il problema della raccomandazione è formalizzato attraverso una funzione d'apprendimento, che utilizza come input sia informazioni di natura "collaborativa" (come i rating) che informazioni più specifiche sul contenuto, e riesce a fornire come output un giudizio di gradimento sul prodotto da parte di uno specifico utente.

2.3 L'importanza dei sistemi di raccomandazione per il marketing one-to-one

Nelle precedenti sezioni è stata fornita un'ampia panoramica sui sistemi di raccomandazione e sulle tre principali tipologie in cui essi si presentano: Collaborative filtering, Content-based filtering e modelli "ibridi" tra le due famiglie precedenti.

In questo paragrafo, invece, sposterò il focus sull'argomento principale dell'intero capitolo, ossia le ragioni che hanno reso i sistemi di raccomandazione uno strumento essenziale, e quasi irrinunciabile, per il marketing one-to-one all'interno di contesti aziendali di grandi dimensioni.

Ma cosa si intende esattamente per "marketing one-to-one"? Questo approccio al marketing, sviluppatosi fortemente negli ultimi tempi, sottintende una forte capacità da parte delle imprese di comprendere al meglio le preferenze dei singoli consumatori, allo scopo di fornire loro una vera e propria offerta personalizzata di beni e servizi che possa soddisfare quanto più possibile le loro esigenze [12]. Lo scopo di tali strategie sta nella volontà, da parte delle aziende, di costruire delle relazioni di lungo periodo con i consumatori, in modo tale da incrementare significativamente il valore economico che questi stessi legami con i clienti hanno, durante il loro intero ciclo di vita [12].

Risulta subito chiaro, dunque, in un tale contesto, il ruolo rivestito dai sistemi di raccomandazione: suggerimenti "diretti" che procedono dall'impresa verso il cliente, utili non solo a favorire eventuali futuri acquisti da parte dell'utente, ma anche e soprattutto a consolidare la relazione instaurata con lo stesso; attraverso questo complesso insieme di tecniche, le imprese possono far sentire la propria "vicinanza" al consumatore, mostrando cura e interesse per le sue preferenze e capacità di favorirne la costituzione di nuove, come mostrato dal meccanismo di "serendipidità" illustrato all'interno della sezione precedente.

Come si è visto, attraverso dei sistemi di raccomandazione di tipo "collaborativo" è possibile sfruttare le possibili convergenze tra gli interessi di utenti simili tra loro per poter costruire dei suggerimenti efficaci: tale meccanismo può permettere al cliente di conoscere ed apprezzare nuovi prodotti, anche di tipologie diverse rispetto agli acquisti consuetudinari; di contro, sistemi di natura "content-based" possono permettere una conoscenza più profonda delle preferenze di ciascun consumatore, consentendo all'impresa di poter strutturare la propria offerta di valore nel modo più accurato e "mirato" possibile.

In conclusione, i sistemi di raccomandazione, siano essi di tipo "Collaborative" o "Content-based", costituiscono una risorsa notevole a disposizione delle imprese nell'era del marketing relazionale, basato su forme sempre più "dirette" di comunicazione con il cliente, permettendo alle aziende di adattare efficientemente e in maniera flessibile le proprie proposte di prodotti o servizi alle rinnovate esigenze di ciascun cliente presente in portafoglio.

Capitolo 3

Le reti neurali artificiali: tipologie e generalità

Nel marzo del 2016, un supercomputer sviluppato da Google DeepMind, AlphaGo, inflisse una storica sconfitta al 18 volte campione mondiale di Go[1] Lee Sedol, in una partita giocata a Seul, nella Corea del Sud [38]. Soltanto un anno più tardi, AlphaGo è riuscito a ripetersi, migliorando al contempo il proprio score precedente, battendo per 3-0 a Wuzhen, in Cina, l'allora numero uno al mondo di Go, Ke Jie [33].
Ma cos'è che ha reso questi eventi tanto eccezionali, specie se comparati alla nota sconfitta di Kasparov[2] di venti anni prima?
Per dare una risposta adeguata a questa domanda, si pensi semplicemente al numero di mosse legali ammissibili in ciascuno dei due giochi: se per gli scacchi si può arrivare a un massimo di 10^{43} [35, p. 92], in Go tale numero cresce esponenzialmente, fino ad arrivare a circa 10^{170} [35, p. 92] mosse consentite (per avere un'idea della grandezza di questa quantità, si immagini che il numero stimato di atomi che compongono l'universo

[1] Gioco da tavolo cinese in cui due giocatori collocano delle pedine di colore bianco e nero su una scacchiera di nome goban, dotata di una griglia 19x19, e il cui scopo è controllare con esse una frazione del goban maggiore di quella dell'avversario.

[2] Riferimento alla famosa sfida a scacchi del maggio 1997 tra il campione russo Garri Kasparov e il supercomputer IBM Deep Blue, vinta clamorosamente da quest'ultimo in sole diciannove mosse.

è pari circa a 10^{80}).

Deep Blue era in grado di "calcolare" la mossa migliore da compiere in ciascun turno ricostruendo un diagramma ad albero che rappresentasse tutte le possibili giocate dei due sfidanti[3] [18, p. 43]; tuttavia, di fronte ad un numero tanto elevato di azioni effettuabili, il sistema perdeva completamente la propria efficacia predittiva, rendendo per tale ragione Go un gioco apparentemente impossibile da "padroneggiare" per una macchina.

Ma in che modo dunque AlphaGo è riuscito a superare questa barriera, battendo l'uomo in un dominio, quello di Go, che sembrava essere un frontiera invalicabile per l'intelligenza artificiale? La risposta ci è fornita direttamente dal campione sconfitto, Ke Jie, al termine della terza partita persa su tre contro il supercomputer ideato da Google: "Lo scorso anno pensavo che il modo di giocare di AlphaGo fosse parecchio simile a quello degli esseri umani, ma oggi penso che giochi come il Dio di Go" [33].

Alla base di AlphaGo, infatti, vi era una rete neurale artificiale [13], un algoritmo che, essendo mutuato dal Machine Learning (di cui ne costituisce una branca a tal punto vasta da essere considerata una scienza a parte, come in seguito si vedrà), ha permesso di migliorare le capacità predittive della macchina attraverso l'"addestramento" della stessa, ossia la somministrazione di ingenti volumi di dati di training, che, nel caso del supercomputer sviluppato da Google, erano costituiti dal quantitativo di partite (più di un milione in un anno) giocate da AlphaGo contro sè stesso [17].

In tal modo la macchina ha registrato un progressivo miglioramento in un lasso di tempo decisamente breve, che le ha permesso di diventare, da giocatore dalle apparenze quasi "umane", un vero e proprio "Dio di Go".

I seguenti paragrafi mirano a fornire al lettore una panoramica esaustiva sulle reti

[3] Più specificamente, Deep Blue utilizzava un algoritmo decisionale detto Minimax [18, p. 43], in cui uno dei due giocatori, detto massimizzante, compie le proprie mosse cercando di ottenere il massimo guadagno possibile, mentre l'altro giocatore, detto minimizzante, effettua le proprie giocate allo scopo di rendere minimo il guadagno del proprio avversario; seguendo tale logica, il computer ricostruiva l'intero albero rappresentante lo spazio delle possibili decisioni dei due giocatori, partendo dalla radice per arrivare ai nodi più alti. Nel caso specifico di Deep Blue, inoltre, all'algoritmo Minimax si affiancava un metodo di ottimizzazione del numero di computazioni necessarie a costruire l'albero decisionale, detto "potatura α-β" [18, p. 43].

neurali artificiali, una famiglia di algoritmi particolarmente complessa e dall'utilizzo estremamente variegato, che include campi come la già citata simulazione ludica, la computer vision[4], il natural language processing[5] e finanche i sistemi di raccomandazione, come si vedrà più avanti nei capitoli conclusivi di questa dissertazione.

3.1 Dalla rete neurale biologica alla rete neurale artificiale

Nell'introduzione al presente capitolo si è osservato come AlphaGo sia riuscito a battere l'uomo in un gioco dove le macchine avevano in precedenza manifestato delle scarse performance, attraverso l'apprendimento automatico risultante dal quantitativo di partite giocate contro sè stesso [17]. Ma in che modo è avvenuto questo processo di apprendimento? Come ha potuto AlphaGo utilizzare la conoscenza maturata "sul campo" per poter migliorare il proprio stile di gioco?
Trattandosi di un'intelligenza artificiale, il supercomputer creato da Google DeepMind ha ovviamente cercato di replicare quanto più possibile i pattern relativi al ragionamento umano, tentando al contempo di simularne il processo di apprendimento.
Nell'uomo, come è risaputo, la conoscenza è acquisita a livello cerebrale attraverso una fitta rete di neuroni, chiamata, per l'appunto, rete neurale (o rete neurale biologica, per distinguerla dalle reti neurali artificiali che su di essa sono basate). Più precisamente, gli stimoli sensoriali sono percepiti dall'uomo attraverso dei ricettori, che li convertono in impulsi elettrici da trasmettere alla rete neurale; infine, la rete converte tali impulsi trasmettendoli agli effettori, che permettono al corpo umano di porre in atto dei comportamenti di risposta agli stimoli [34, p. 6].

[4]Branca dell'intelligenza artificiale che studia le modalità di elaborazione ed interpretazione di dati visivi da parte delle macchine.

[5]Campo dell'intelligenza artificiale che studia le tecniche di elaborazione del linguaggio naturale da parte dei computer.

Il neurone è la struttura basilare del sistema nervoso umano [21, p. 357] ed è composto da tre elementi principali:

- I dendriti, che ricevono gli impulsi elettrici dagli altri neuroni e li trasmettono al corpo cellulare [21, p. 357];
- Il corpo cellulare, che contiene il nucleo del neurone ed è il fulcro dell'attività cerebrale [21, p. 357];
- L'assone, che riceve i segnali dal corpo cellulare e li trasmette agli altri neuroni della rete, attraverso numerose ramificazioni [21, p. 357].

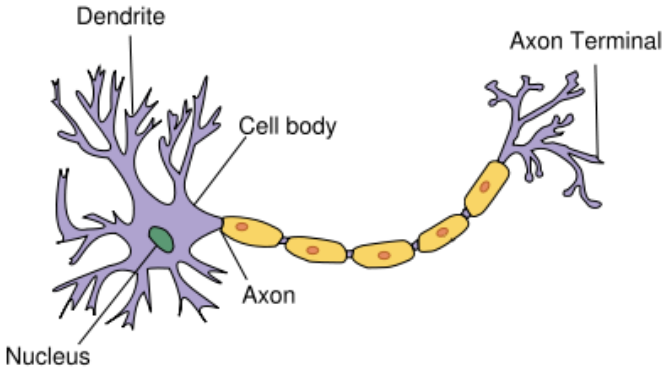

Figura 3.1. Struttura di un neurone biologico.

L'impulso elettrico, ricevuto da ciascun neurone attraverso i dendriti (la cui numerosità è dovuta al grande quantitativo di interconnessioni presenti tra i singoli elementi della rete neurale), è successivamente "propagato" all'interno della rete sfruttando la capacità di resistenza dell'assone a tali stimoli [34, p. 7]; l'intero neurone biologico, complessivamente, raggiunge le dimensioni di 100 micrometri[6] [34, p. 7].
La figura in basso evidenzia chiaramente quanto detto in precedenza: il segnale è propagato da neurone a neurone attraverso le connessioni tra le terminazioni assoniche dell'emittente e i dendriti della cellula destinataria.

[6]Un micrometro, la cui unità di misura è μm, corrisponde all'incirca a $1 \cdot 10^{-6}$ metri.

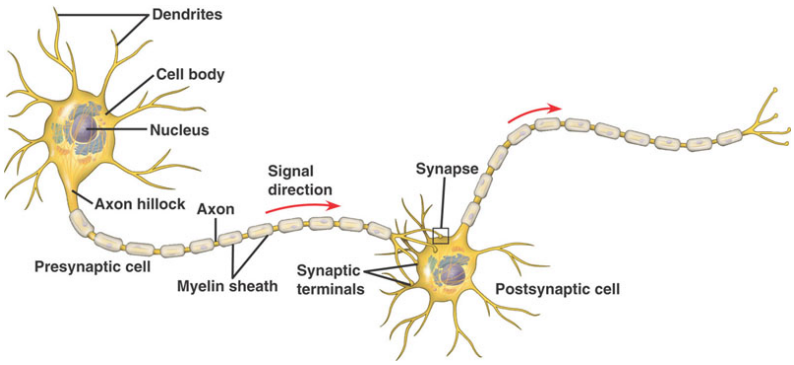

Figura 3.2. Modalità di propagazione dell'impulso elettrico tra due neuroni.

La struttura delle reti neurali biologiche ha fortemente influenzato, come è facilmente intuibile, la forma delle rispettive controparti "artificiali".
Come vedremo dettagliatamente nella prossima sezione, un neurone artificiale si compone essenzialmente di tre elementi [34, p. 10], che costituiscono la base di ciascuna struttura reticolare che sarà analizzata in questa tesi:

- Una serie di legami connettivi che collegano il neurone k a un determinato numero di input, caratterizzati da un "peso" ω_{kj} che esprime l'intensità del j-esimo legame tra l'input x_j e il neurone in questione [34, p. 10];
- Un elemento di sommatoria dei diversi segnali di input, ponderati per i rispettivi pesi [34, p. 10];
- Una funzione di attivazione, che prende in input il risultato della sommatoria precedente e, sulla base di quest'ultimo, determina l'"accensione" o meno del neurone e la propagazione dell'output all'interno della rete [34, p. 10].

Come facilmente intuibile, i legami connettivi in entrata del neurone artificiale non rappresentano altro che l'equivalente dei dendriti del neurone biologico, in quanto trasmettono un impulso, di intensità variabile (legata al "peso" del legame in questione), al nucleo centrale [21, p. 358]; la successiva operazione di sommatoria degli input, propagata poi attraverso una funzione di attivazione, non è altro che la simulazione di quanto avviene nel corpo cellulare del neurone, dove la forza del segnale elettrico determina o

meno l'"attivazione" dello stesso [21, p. 358]. Infine, nel caso in cui l'attivazione si sia verificata con successo, il segnale viene trasmesso agli altri neuroni della rete attraverso nuovi legami connettivi, che stavolta rappresentano l'assone del neurone artificiale [21, p. 358].

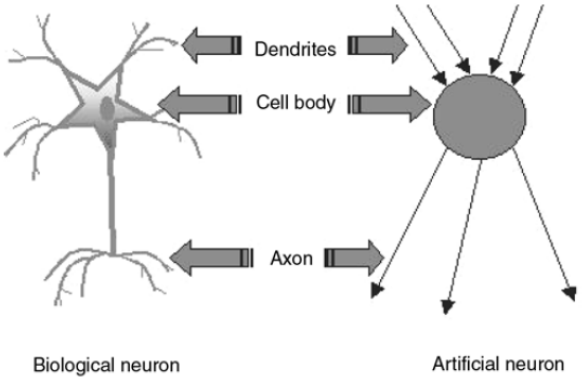

Figura 3.3. Analogie tra un neurone biologico ed un neurone artificiale [21, p. 358].

Nella prossima sezione inizierà la trattazione vera e propria delle reti neurali artificiali, discutendone dettagliatamente un primo approccio noto come "percettrone di Rosenblatt"; nei successivi paragrafi, invece, saranno illustrati modelli dotati di maggiore complessità, come il percettrone multistrato, le reti neurali ricorrenti e le reti convoluzionali.

3.2 Una prima forma di rete neurale artificiale: il percettrone

Nonostante rappresentino ad oggi uno degli strumenti più avanzati della tecnologia moderna, le reti neurali artificiali non hanno certo avuto origine in tempi recenti: risale infatti al 1958, con l'articolo "The perceptron: a probabilistic model for information storage and organization in the brain", ad opera dello psicologo americano Frank Rosenblatt [16], la prima "formalizzazione" del concetto di rete neurale artificiale, attraverso un modello di notevole semplicità definito "percettrone" [23, cap.1].

Il percettrone proposto da Rosenblatt è, in estrema sostanza, un algoritmo di classificazione[7] che, partendo da un numero n di input $x_1, x_2, ..., x_n$, produce un singolo output binario (0 o 1 a seconda dell'appartenenza o meno delle osservazioni alla classe in esame) [23, cap.1].

A ciascuno degli n input è attribuito un peso $\omega_1, \omega_2, ..., \omega_n$ [23, cap.1], in modo tale che l'output finale del percettrone possa essere determinato dal confronto tra la somma pesata degli input, $\sum_n x_n \cdot \omega_n$, e uno specifico valore soglia t, come evidenzia l'espressione seguente [23, cap.1]:

$$\text{Output} = \begin{cases} 0, & \text{se } \sum_n x_n \cdot \omega_n \leq t \\ 1, & \text{se } \sum_n x_n \cdot \omega_n > t \end{cases} \tag{3.1}$$

Ma in cosa consistono esattamente i pesi attribuiti a ciascuno degli n input? Semplicemente, essi rappresentano l'importanza relativa che ciascun input riveste per la formazione di un determinato output [23, cap.1].

Si consideri ad esempio una generica azienda, la quale ha bisogno di determinare se lanciare o meno un nuovo prodotto sul mercato: il successo di questa iniziativa dipenderà, come è facilmente intuibile, da diversi fattori, come ad esempio la consistenza del mercato, la presenza di eventuali competitors e l'elasticità della domanda.

Naturalmente non tutti questi elementi contribuiscono in equal modo a determinare la decisione finale dell'azienda, per via della loro differente importanza: è in tal senso, dunque, che a ciascuno di questi input è attribuito uno specifico peso.

[7] Nella teoria dell'apprendimento supervisionato, gli algoritmi di classificazione rappresentano quella famiglia di tecniche predittive (di cui fanno parte la Regressione logistica, gli Alberi di decisione, il Naïve Bayes e molti altri strumenti) che, partendo da un dato numero di input, ottengono come proprio output non un numero reale intepretabile come tale (a differenza dei c.d. "algoritmi di regressione"), ma una classe, ossia una categoria di appartenenza per le osservazioni (ad esempio "spam" o "non spam", "vero" o "falso", "acquisto" o "non acquisto").

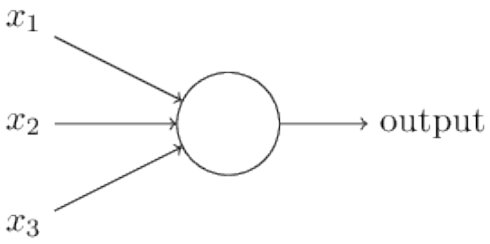

Figura 3.4. Struttura di un percettrone [23, cap.1].

Nelle applicazioni più recenti, capita frequetemente di ritrovare il valore soglia t "riportato" come termine sinistro della disequazione (3.1), in modo tale da rendere più chiara la lettura dell'intera espressione [23, cap.1]; il nuovo parametro risultante da questa operazione, definito "bias" del percettrone, è convenzionalmente indicato con b ed è pari, come anticipato in precedenza, a -t [23, cap.1].
Alla luce di ciò, l'espressione (3.1) può essere reinterpretata nel modo seguente, inserendo il bias come parametro sostitutivo del valore soglia [23, cap.1]:

$$\text{Output} = \begin{cases} 0, & \text{se } \sum_n x_n \cdot \omega_n + b \leq 0 \\ 1, & \text{se } \sum_n x_n \cdot \omega_n + b > 0 \end{cases} \tag{3.2}$$

Le ragioni che rendono tendenzialmente preferibile l'utilizzo del bias come parametro del modello, in alternativa al valore t, sono riconducibili alla sua notevole semplicità interpretativa: esso indica, infatti, il grado di facilità con cui un neurone può "accendersi" restituendo come output 1 [23, cap.1], in quanto, per valori molto alti di b, risulterebbe estremamente difficile che l'intera espressione $\sum_n x_n \cdot \omega_n + b$ possa assumere valori minori o uguali a 0 [23, cap.1].
In tal senso, l'espressione precedente può essere considerata la "funzione di attivazione" del percettrone [34, p. 13], in quanto determina le condizioni necessarie affinchè il neurone artificiale si "attivi", producendo come ouput 1, o non lo faccia, restituendo di contro il valore 0.
Seguendo tale ragionamento, la (3.2) può essere riscritta come segue [34, p. 13]:

$$\text{Output} = \theta(x_n \cdot \omega_n + b) \tag{3.3}$$

Dove θ esprime la funzione di attivazione della rete neurale, che nel caso del percettrone è pari a [34, p. 13]:

$$\theta = \begin{cases} 0, & \text{se } \sum_n x_n \cdot \omega_n + b \leq 0 \\ 1, & \text{se } \sum_n x_n \cdot \omega_n + b > 0 \end{cases} \tag{3.4}$$

Questa tipologia di funzione di attivazione è anche nota come "funzione di Heaviside" [34, p. 13] ed ha una caratteristica forma "a gradino": assume un valore pari a 0 in corrispondenza di grandezze della variabile indipendente minori o uguali a 0, mentre in caso contrario vale 1.

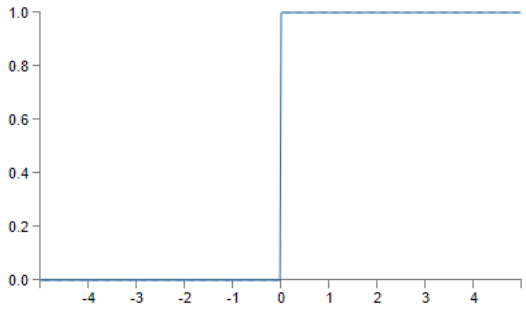

Figura 3.5. Funzione di attivazione di Heaviside [23, cap.1].

Uno dei problemi connessi all'utilizzo di una funzione di attivazione della tipologia descritta in precedenza è, tuttavia, la forte sensibilità del percettrone a variazioni anche minimali nei pesi e nel bias [23, cap.1].
Ciò è dovuto alla particolare struttura della funzione di Heaviside, la quale è costruita allo scopo di predire unicamente due tipologie di output: uno 0 per indicare la non appartenenza delle osservazioni alla classe d'interesse, e un 1 nel caso si verifichi la condizione opposta; per tale ragione potrebbe capitare che variazioni anche sufficientemente piccole nei parametri del modello determinino un ribaltamento totale nel valore di output, che potrebbe passare da 0 a 1 e viceversa [23, cap.1].

Inoltre, un ulteriore criticità connessa all'utilizzo della funzione di attivazione di Heaviside è la sua non differenziabilità [34, p. 13]; assumendo infatti due soli valori, 0 e 1, la funzione "a gradino" risulta essere discreta e non continua, fatto questo che potrebbe determinare non pochi problemi[8] nel successivo addestramento del modello [34, p. 14]. Per ovviare a queste complicazioni, in tempi più recenti è stata introdotta una nuova tipologia di rete neurale, che costituisce una variante del percettrone originario: il neurone "sigmoideo" [23, cap.1].

I neuroni sigmoidei sono caratterizzati dall'utilizzo di una funzione di attivazione diversa da quella di Heaviside, nota come "funzione sigmoidea", per la quale l'output finale del modello può essere calcolato come [23, cap.1]:

$$\sigma = \frac{1}{1 + e^{-\sum_n x_n \cdot \omega_n + b}} \tag{3.5}$$

In questo caso, la funzione assume qualunque valore reale tra 0 e 1 (risultando per tale ragione continua e differenziabile) e, come la funzione di Heaviside, tende a predire il valore 1 all'aumentare di $\sum_n x_n \cdot \omega_n + b$ (in quel caso, infatti, il termine a denominatore diminuisce) [23, cap.1].

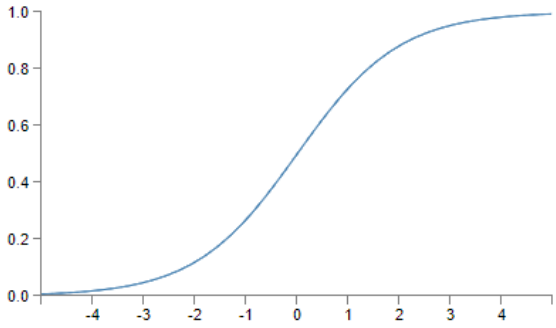

Figura 3.6. Funzione di attivazione sigmoidea [23, cap.1].

[8] Come si vedrà nel capitolo 4, la differenziabilità della funzione di attivazione è una condizione necessaria per l'utilizzo dell'algoritmo di backpropagation.

L'utilizzo di questa tipologia di neurone permette inoltre di "superare" il problema, tipico del percettrone, della forte suscettibilità dell'output a variazioni nei pesi e nei bias: la funzione sigmoidea, infatti, restituisce un numero reale compreso tra 0 e 1, il quale può essere sia utilizzato in quanto tale, sia confrontato a un valore soglia t (similmente a quanto avveniva con il modello di Rosenblatt), in caso in cui la situazione in esame richieda una classificazione delle osservazioni [23, cap.1].

3.3 Dallo "shallow learning" al "deep learning": il percettrone multistrato

Ambedue i modelli di rete neurale discussi sinora, ossia il percettrone di Rosenblatt e la sua variante "sigmoidea", presentano una notevole semplicità concettuale e strutturale, in quanto composti da due soli "strati": uno, contenente i valori di input $x_1, x_2, ..., x_n$ e un secondo, contenente il valore di output corrispondente all'attivazione $f(x_n \cdot \omega_n + b)$ [23, cap.1].

Tuttavia esistono modelli di natura più articolata, dotati di una complessità maggiore e di un numero decisamente più ampio di "strati", in modo tale da simulare quanto più possibile ciò che nella realtà avviene per le reti neurali biologiche: un esempio comune è il c.d. "percettrone multistrato" [23, cap.1], una tipologia di rete neurale la quale, pur condividendo il nome con l'algoritmo proposto da Rosenblatt, è in realtà una variante del neurone "sigmoideo" che presenta strati intermedi tra quello di input e quello di output [23, cap.1].

Più precisamente, gli strati mediani del percettrone multistrato (così come di ogni altra architettura di rete neurale maggiormente complessa) sono definiti "strati nascosti" [23, cap.1] e in essi è applicata, similmente a quanto avviene per lo strato di input, una funzione di attivazione che trasforma i dati in ingresso in valori utilizzabili dai successivi strati della rete, o direttamente dallo strato finale di output in caso di reti dalla struttura non eccessivamente elaborata [23, cap.1].

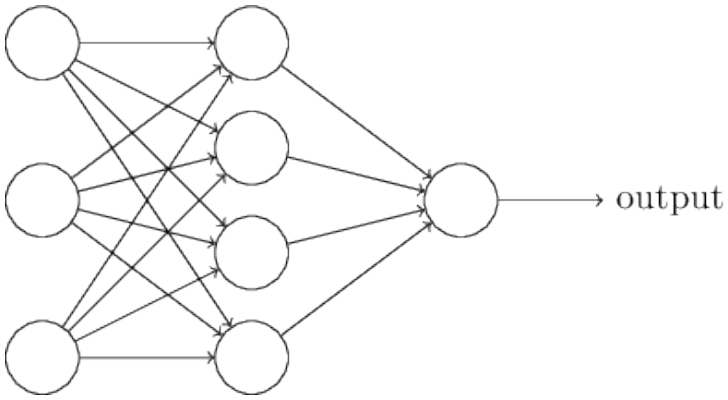

Figura 3.7. Struttura di un percettrone multistrato con un solo strato nascosto [23, cap.1].

Il percettrone multistrato rappresenta inoltre la forma meno complessa[9] di rete neurale "feed-forward", ossia quella tipologia di modelli in cui le informazioni sono propagate nella rete soltanto in avanti [23, cap.1], e non vi è presenza di cicli o ricorsioni (come avviene invece per le reti "feed-back", che saranno discusse alla fine di questo capitolo). Il percettrone, come già anticipato, può avere più di uno strato nascosto [23, cap.1]; ovviamente, all'aumentare del numero degli strati intermedi, incrementa notevolmente anche la complessità del modello e il numero di trasformazioni dei dati iniziali richieste per giungere all'output finale [23, cap.1].

Per tale ragione, reti neurali dotate di un'architettura simile (ossia comprendente due o più strati nascosti) sono anche definite "profonde", e l'intera disciplina di studio cui esse sono collegate è nota con il nome di "deep learning" [23, cap.1].

Reti neurali con diversi strati nascosti permettono, come si può facilmente immaginare, di ottenere dei risultati finali maggiormente accurati in virtù delle numerose trasformazioni dei dati iniziali che vengono compiute ai livelli intermedi; per questo motivo, tali strumenti, indubbiamente dotati di notevole complessità, sono spesso preferiti a modelli più "snelli" (detti anche reti neurali "shallow"[10], ossia poco profonde) in diverse

[9] Se si escludono ovviamente le architetture a due soli strati che sono state descritte nella sezione precedente.

[10] Il termine "shallow learning" è stato coniato proprio per designare il campo di studio delle reti neurali con non più di uno strato nascosto, ed è spesso utilizzato in contrapposizione al concetto di

applicazioni reali[11] [23, cap.1].

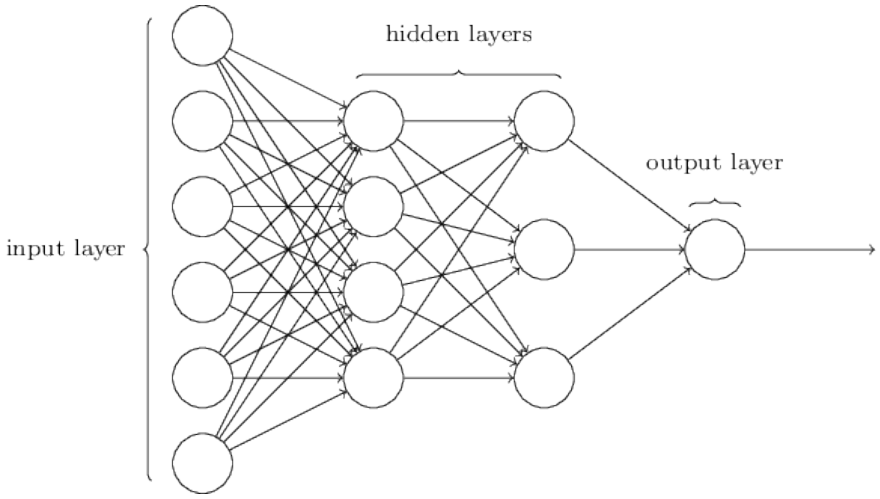

Figura 3.8. Struttura di un percettrone multistrato con più strati nascosti [23, cap.1].

Allo scopo di fornire al lettore un esempio chiaro ed efficace del funzionamento di un percettrone multistrato, ne citerò un'applicazione empirica tratta dal manuale "Neural Networks and Deep Learning" di Michael Nielsen [23, cap.1], in cui tale strumento è utilizzato per costruire un classificatore di immagini, riportanti delle cifre scritte a mano, estrapolate dal noto dataset MNIST[12].

Il compito del percettrone multistrato in questione è determinare, partendo dal numero scritto a mano in ciascuna delle immagini contenenti nel dataset, la corrispondente cifra da 0 a 9 [23, cap.1]; più specificamente, il percettrone multistrato utilizzato nell'esempio è di tipo "shallow", costituito da uno strato di input, contenente ben 784 neuroni (essendo le immagini di dimensione 28x28x1[13]), uno strato nascosto, contenente 15

"deep learning".

[11] Come si vedrà nell'ultimo capitolo, anche il mio lavoro di ricerca confermerà questo assunto, mostrando un errore predittivo minore per una rete neurale "profonda" rispetto ad una controparte con un unico strato nascosto.

[12] Acronimo di Modified National Institute of Standards and Technology, è un database contenente circa 70.000 immagini ritraenti numeri scritti a mano, in formato 28x28x1, di cui 60.000 sono usate come training set e le restanti per la fase di testing.

[13] Tali misure si riferiscono alle dimensioni della singola immagine, che ha altezza e larghezza pari a

neuroni, e uno strato finale, composto di 10 neuroni, uno per ogni possibile esito della classificazione [23, cap.1].

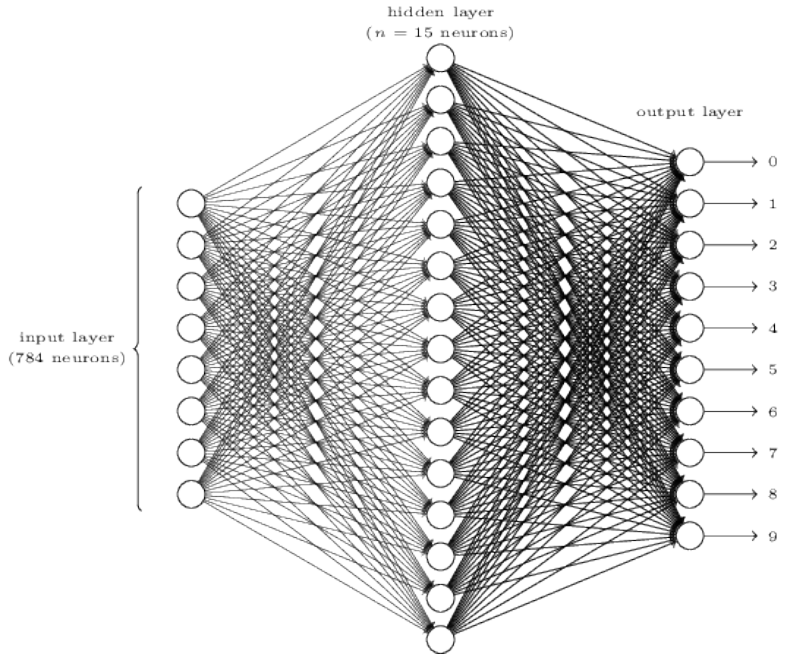

Figura 3.9. Il percettrone multistrato utilizzato nell'esempio [23, cap.1].

Ma per quale motivo l'autore ha utilizzato 15 neuroni per costruire lo strato nascosto? Non sarebbe stato sufficiente inserirne una quantità minore, diminuendo il numero di computazioni necessarie ad ottenere l'output finale?
La ragione di questa scelta sta nella funzione che ciascun neurone dello strato nascosto possiede: aggiungere informazioni ulteriori rispetto ai dati "grezzi" contenuti nello strato di input, per consentire delle predizioni maggiormente accurate [23, cap.1].
Per esplicare ancor meglio quest'ultimo concetto, l'autore del manuale fornisce un ulteriore esempio: si pensi a un caso in cui il percettrone multistrato debba determinare se

28 pixel e profondità pari al numero di canali, ossia di pigmenti basilari che ne determinano il colore, che nel caso in esame è 1. Nello specifico, le immagini contenute nel dataset MNIST hanno un unico canale perchè sono in scala di grigi, mentre le comuni immagini a colori possiedono in genere tre canali, corrispondenti alle tonalità primarie Rosso, Verde e Blu (RGB).

una specifica immagine contenga o meno uno zero [23, cap.1], e il primo neurone dello strato nascosto contenga una funzione di attivazione tale da permettergli di capire se sia presente all'interno del dato di input il seguente carattere:

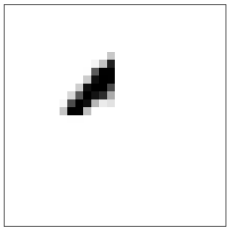

Figura 3.10. Primo carattere utilizzato nell'esempio [23, cap.1].

Risulta chiaro che tale figura non sia sufficiente per permettere al modello di effettuare una predizione corretta, in quanto una linea di quella tipologia potrebbe essere contenuta anche in numeri diversi dallo 0 [23, cap.1].
Tuttavia, si ipotizzi in maniera del tutto analoga che il secondo, terzo e quarto neurone dello strato nascosto contengano una funzione di attivazione dello stesso tipo della precedente, ma deputata a cogliere nell'immagine iniziale le seguenti forme [23, cap.1]:

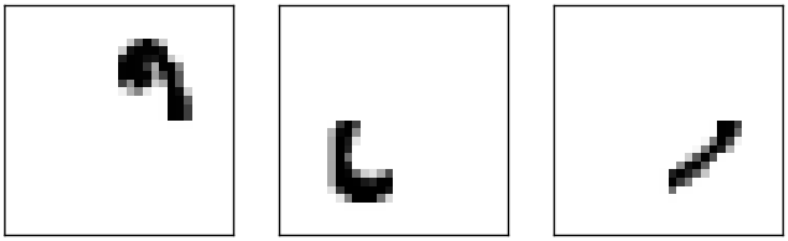

Figura 3.11. Secondo, terzo e quarto carattere utilizzati nell'esempio [23, cap.1].

In questo caso le informazioni aggiuntive rappresentano un evidente miglioramento rispetto allo stato iniziale, e possono certamente incrementare la capacità predittiva del modello [23, cap.1]. Di conseguenza, appare chiaro il ruolo che ciascun neurone all'interno dello strato nascosto svolge nella determinazione dell'output finale, consentendo al percettrone multistrato di aggiungere sempre nuovi tasselli ai propri dati iniziali,

guadagnando al contempo una notevole accuratezza nel determinare le classi di appartenenza delle osservazioni in esame [23, cap.1].
Il percettrone multistrato rappresenta indubbiamente un efficace punto di partenza per lo studio delle reti neurali "profonde" e del deep learning, pur non esaurendo la vasta gamma di strumenti che afferiscono a tali ambiti: nella prossima sezione, infatti, verrà ultimata la panoramica sulle reti "feed-forward", discutendone una delle forme più complesse e al contempo utilizzate, nota come "rete neurale convoluzionale"; in seguito, nella sezione conclusiva di questo capitolo, saranno introdotte le reti di tipo "feed-back" attraverso l'analisi dettagliata del modello di rete neurale ricorrente.

3.4 Le reti neurali convoluzionali

Nel precedente paragrafo si è mostrato, attraverso un esempio pratico, come le reti neurali artificiali possano essere utilizzate per la risoluzione di problemi complessi, come la classificazione di un'immagine digitale: tuttavia, come si è avuto modo di notare, attraverso il percettrone multistrato si è assolto a tale compito utilizzando ben 784 neuroni nello strato di input, uno per ogni pixel di cui l'immagine si compone.
Appare evidente che questo singolare comportamento in presenza di dati grafici, per loro natura tridimensionali, rende l'uso del percettrone multistrato estremamente inefficiente nella risoluzione di questa tipologia di problemi [22, pp. 2-3]; considerando anche le ridotte dimensioni delle immagini contenute nel dataset MNIST, il numero di computazioni necessarie ad arrivare allo strato finale di output incrementerebbe in maniera drastica in presenza di dati di grandezza superiore, come immagini 64x64x1 (che richiederebbero ben 4096 neuroni) o rappresentazioni grafiche multi-canale in formato RGB[14] [22, pp. 2-3]. Per tale ragione, nella quasi totalità dei casi, si fa uso di reti neurali dotate di architetture diverse dal percettrone multistrato nell'ambito della ricognizione grafica e della "computer vision", come le c.d. "reti convoluzionali" [23, cap.6].

[14]Che, nel caso delle immagini 64x64, innalzerebbe il numero di neuroni iniziali a 12288.

Una rete neurale convoluzionale è un modello con una struttura decisamente più complessa rispetto al semplice percettrone, in quanto, come vedremo, ciascun suo strato possiede delle caratteristiche "uniche" ed applica trasformazioni estremamente diverse da quelle incontrate negli algoritmi precedenti; innanzitutto, lo stesso strato di input di una rete convoluzionale è strutturato in modo tale da rispettarne la costituzione "spaziale", ragion per cui, rifacendoci all'esempio dell'immagine di dimensione 28x28x1, essa sarà trattata non come una sequenza di 784 input indipendenti, ma come un "quadrato" dove i pixel seguono la propria disposizione reale, come quello in figura [23, cap.6]:

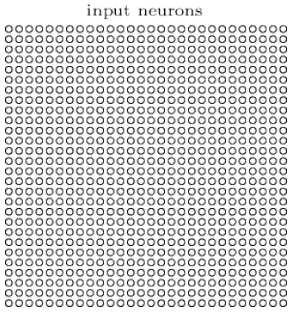

Figura 3.12. Strato di input di una rete neurale convoluzionale [23, cap.6].

Nelle reti convoluzionali, a differenza di quanto visto in precedenza per il percettrone multistrato, lo strato di input e il successivo strato nascosto non sono "fully connected" [23, cap.6]: ciò significa che, all'interno di questa tipologia di modelli, ciascun neurone dello strato nascosto è "connesso" soltanto con un sottospazio dello strato iniziale, definito anche "spazio locale ricettivo" [23, cap.6].

L'insieme dei pesi da applicare allo strato di input viene utilizzato per costruire una piccola matrice, delle stesse dimensioni di uno spazio locale ricettivo, nota come "filtro" o "kernel" [23, cap.6]. Una volta determinati i pesi, il kernel viene utilizzato per effettuare un'operazione matematica sullo strato di input nota come "convoluzione", in cui si determina ciascun neurone dello strato nascosto come prodotto scalare tra gli elementi del filtro e quelli di ognuno degli spazi locali ricettivi di cui si compone l'immagine iniziale [22, p. 4].

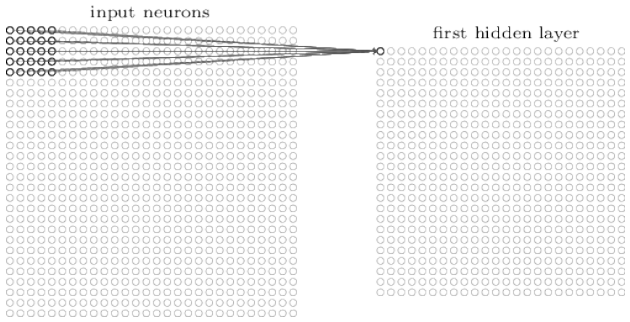

Figura 3.13. Creazione di uno strato nascosto attraverso un'operazione di convoluzione tra gli spazi locali ricettivi e il kernel [23, cap.6].

Il pesi contenuti all'interno di un kernel sono rappresentativi di una determinata caratteristica dell'immagine iniziale [23, cap.6], e lo strato nascosto determinato attraverso l'operazione di convoluzione osservata in precedenza descrive esattamente la stessa "feature", localizzata in diverse posizioni della rappresentazione originale [23, cap.6].
Per tale ragione, tale strato è anche definito "mappa delle caratteristiche" (feature map) e, attraverso l'utilizzo di altri kernel, è possibile generare nuove mappe dall'immagine di input effettuando delle convoluzioni ulteriori tra i filtri e gli spazi locali ricettivi, come mostrato nella figura seguente [23, cap.6]:

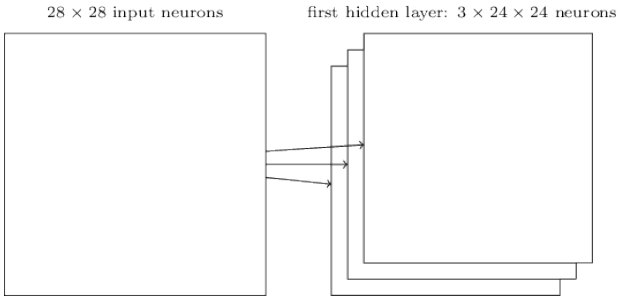

Figura 3.14. Creazione di più "mappe" attraverso nuove operazioni di convoluzione [23, cap.6].

Questo strato intermedio, costituito da più "feature maps", è noto anche come "strato convoluzionale", dal nome dell'operazione matematica utilizzata per generarlo [23, cap.6].
In genere, per permettere una notevole semplificazione delle computazioni necessarie

a pervenire allo strato finale, è applicata allo strato convoluzionale una funzione di attivazione nota come "Rectified Linear", o ReLu [22, p. 4-5], la quale prende in input il risultato z della convoluzione per ciascuno spazio locale ricettivo, generando come output il nuovo neurone nascosto h [23, cap.3]:

$$h = \max(0, z) \tag{3.6}$$

Utilizzando la ReLu, ogni possibile elemento negativo delle "feature maps" è convertito in 0, lasciando al contempo inalterati i valori positivi generati dalla convoluzione [22, p. 4-5]; in questo modo, si ottiene un sostanziale incremento della velocità di apprendimento della rete e di propagazione dei risultati negli strati successivi [22, p. 5].

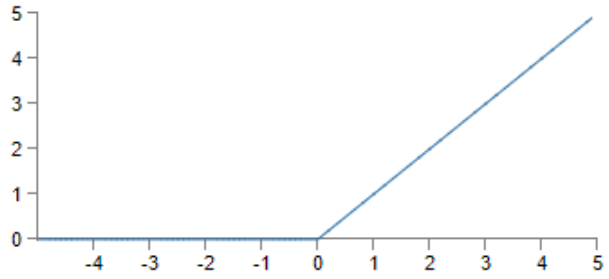

Figura 3.15. Funzione di attivazione ReLu [23, cap.3].

Generalmente, all'interno di reti dotate di questa tipologia di architettura, si fa seguire, allo strato convoluzionale descritto in precedenza, un ulteriore livello nascosto noto anche come "strato di pooling" [23, cap.6]: quest'ultimo ha unicamente lo scopo di sintetizzare ancor più le informazioni propagate nei livelli precedenti della rete, attraverso una procedura definita "max pooling" [23, cap.6], in cui le "feature maps" costituenti il primo strato nascosto vengono suddivise in piccole "regioni" (ad esempio, aree 2x2, 3x3, 4x4 etc.), delle quali soltanto il valore più alto viene utilizzato come "neurone" per

costituire il successivo strato intermedio [23, cap.6].

In questo modo, come mostra la figura in basso, si verifica una notevole riduzione dimensionale delle informazioni che saranno utilizzate dai livelli successivi della rete [23, cap.6]:

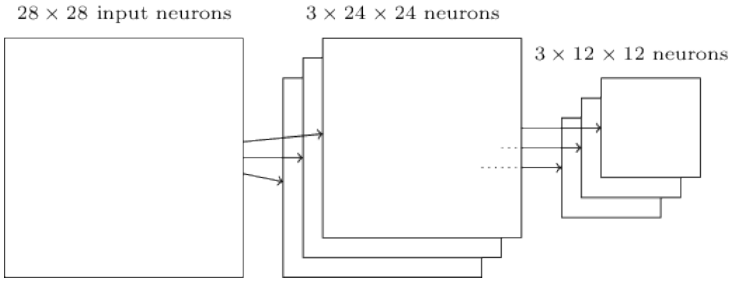

Figura 3.16. Creazione di uno strato di pooling attraverso un operazione di "max pooling" sullo strato convoluzionale [23, cap.6].

All'interno di una rete neurale di questo tipo, sono solitamente utilizzate diverse successioni di strati convoluzionali (ai quali viene poi applicata una funzione di attivazione come la ReLu descritta in precedenza) e strati di pooling, prima di pervenire allo strato finale di output [23, cap.6]. Quest'ultimo è anche definito strato "fully connected" [23, cap.6], in quanto ciascun elemento dello strato di pooling precedente è collegato ad ogni singolo neurone che lo compone [23, cap.6].

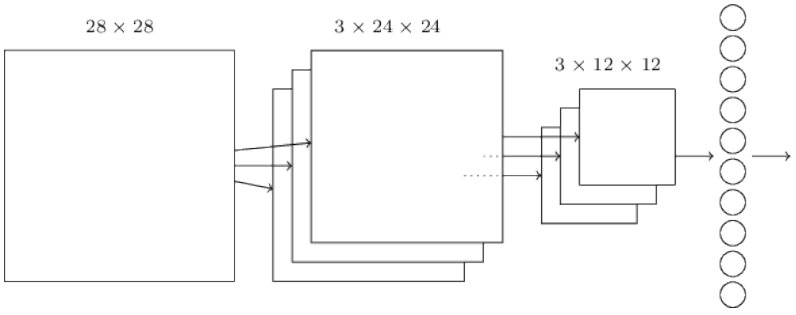

Figura 3.17. Aspetto finale di una rete neurale convoluzionale dotata di uno strato di output "fully connected" [23, cap.6].

Come si può notare dall'immagine precedente, generalmente l'output di una rete convoluzionale è uno strato composto da una sequenza di neuroni, ciascuno rappresentante una possibile classe di appartenenza dell'immagine (ad esempio, nell'applicazione pratica vista in precedenza per il dataset MNIST, vi erano 10 possibili categorie, corrispondenti a una sequenza di numeri da 0 a 9) [23, cap.6].
Per tale ragione, nello strato finale "fully connected" di una rete convoluzionale è spesso utilizzata una versione "modificata" della funzione sigmoidea[15], nota come "softmax" [1, p. 4], particolarmente adeguata in caso di problemi di classificazione in cui il possibile output sia composto da una molteplicità di categorie [23, cap.3].
Nello specifico, la funzione softmax prende in input ciascun neurone dello strato intermedio di pooling, restituendo come output un numero j di attivazioni, ciascuna delle quali è pari a [23, cap.3]:

$$a_j^l = \frac{e^{z_j^l}}{\sum_j e^{z_j^l}} \tag{3.7}$$

Dove per a_j^l s'intende la j-esima attivazione all'interno dell' l-esimo strato [23, cap.2]; z invece corrisponde corrisponde alla somma pesata degli input in quello strato della rete, ossia $\sum_j \omega_j^l \cdot a_j^{l-1} + b_j^l$ [23, cap.3].
Nello specifico, ciò che rende la funzione softmax adatta a problemi di apprendimento automatico in cui vi siano j possibili output finali, è la condizione per cui [23, cap.3]:

$$\sum_j a_j^l = 1 \tag{3.8}$$

Infatti, ciò garantisce che ciascuno dei j output della rete convoluzionale sia un numero reale compreso tra 0 e 1, permettendo, nella quasi totalità delle applicazioni reali, di classificare come categoria di appartenenza dell'immagine di input semplicemente quella cui corrisponde il valore finale più elevato [23, cap.3].

[15] La quale può essere utilizzata solo in caso in cui la classe di appartenenza delle osservazioni di input sia unica, predicendo un solo numero reale compreso tra 0 e 1.

3.5 Modelli "feed-back" e reti neurali ricorrenti

Tutti i modelli di rete neurale analizzati nelle sezioni precedenti, da quelli dotati di architettura più semplice e "snella" (percettroni, neuroni sigmoidei) a quelli maggiormente complessi e articolati (percettroni multistrato e reti neurali convoluzionali), sono accomunati dal fatto di essere reti "feed-forward", ossia strutturate in modo tale che le informazioni siano propagate unicamente in "avanti", partendo dallo strato iniziale per arrivare a quello finale di output [23, cap.6].

Nelle reti neurali "feed-forward", in estrema sintesi, gli input correnti ad un dato livello delle stesse determinano tutte le attivazioni che si verificano negli strati immediatamente successivi [23, cap.3]; se ciò, tuttavia, può rispecchiare pienamente il comportamento di alcune strutture di dati, non permette al contempo di modellare tramite questo tipo di reti quelle tipologie di informazioni che sono, per loro natura, di forma sequenziale [23, cap.6].

Ad esempio, può verificarsi che l'attivazione dei neuroni in uno strato nascosto possa essere influenzata non solo dagli input correnti, ma anche da quelli antecedenti, più prossimi allo strato iniziale [23, cap.6]. Si pensi ad esempio ad un comune problema di comprensione testuale: le due frasi "X è più alto di Y" e "Y è più alto di X" sono assolutamente equivalenti per scelta delle parole usate, ma la sequenza con cui esse si presentano è fondamentale per interpretarne correttamente il significato.

Per far fronte a queste evidenti criticità, sono state introdotte delle tipologie di reti neurali alternative rispetto ai modelli osservati finora, dette anche reti "feed-back" o "reti neurali ricorrenti", strutturate in modo tale da elaborare dati di natura sequenziale, cogliendone e analizzandone le dinamiche temporali alla base [19, p. 367].

Ma qual è esattamente la struttura di una rete neurale "feed-back"?

Si consideri con $s^{(t)}$ lo stato delle computazioni al tempo corrente t[16]: all'interno di

[16]Il concetto di "tempo" in questo caso non è da interpretarsi in senso letterale, ma come sinonimo di "posizione" all'interno di una sequenza ordinata di dati [19, p. 368].

una rete neurale ricorrente, esso può essere rappresentato come [19, p. 369]:

$$s^{(t)} = f(s^{(t-1)}, x^{(t)}, \tau) \tag{3.9}$$

Dove, rispettivamente, $s^{(t-1)}$ è lo stato al tempo precedente, $x^{(t)}$ l'input corrente al tempo t e τ la funzione di attivazione usata in quello strato intermedio della rete neurale.
In questa tipologia di rete, come evidentemente mostrato dalla (3.9), lo stato corrente non dipende unicamente dall'input in t, $x^{(t)}$, e dalla forma dell'attivazione τ usata in quel determinato livello, ma anche da quanto si è verificato in precedenza, rappresentato dallo stato in t-1 [19, p. 369]; inoltre, essendo la formula precedente valida per ogni valore del tempo t, sarà valida anche la condizione per cui [19, p. 369]:

$$s^{(t-1)} = f(s^{(t-2)}, x^{(t-1)}, \tau) \tag{3.10}$$

Da cui è possibile riscrivere la (3.9) come segue [19, p. 369]:

$$s^{(t)} = f(f(s^{(t-2)}; x^{(t-1)}; \tau), x^{(t)}, \tau) \tag{3.11}$$

Da questa prospettiva, appare decisamente chiara la ragione per cui reti neurali di questa tipologia sono definite "ricorrenti": ciascuno stato può essere espresso in funzione della sequenza precedente, dando all'intera struttura una forma "ciclica" e ricorsiva.
Utilizzando una notazione più adatta alla trattazione in esame, si può sostituire a $s^{(t)}$ lo strato nascosto al tempo corrente $h^{(t)}$, per cui la (3.9) può essere reinterpretata nel modo seguente [19, p. 370]:

$$h^{(t)} = f(h^{(t-1)}, x^{(t)}, \tau) \tag{3.12}$$

In questo caso, lo strato nascosto $h^{(t)}$ può essere considerato come funzione dell'attivazione precedente, oltre che dell'input corrente e della funzione τ [19, p. 370]; inoltre, come anticipato in precedenza, poichè tale condizione vale anche per gli strati precedenti, sarà possibile rappresentare la (3.12) anche in una forma maggiormente esplicita,

come segue [19, p. 370]:

$$h^{(t)} = f(f(h^{(t-2)}; x^{(t-1)}; \tau), x^{(t)}, \tau) \tag{3.13}$$

In letteratura, si fa riferimento alla forma assunta dalla rete neurale ricorrente nella (3.12) con l'espressione di "rete ricorrente ripiegata" (folded), per evidenziare la natura "ciclica" e ricorsiva delle informazioni in essa contenute [19, p. 371]; al contrario, la rete mostrata nell'equazione (3.13) è anche definita "rete ricorrente dispiegata" (unfolded), in quanto, pur rappresentando le stesse informazioni della prima, le evidenzia in una forma maggiormente esplicita [19, p. 371].

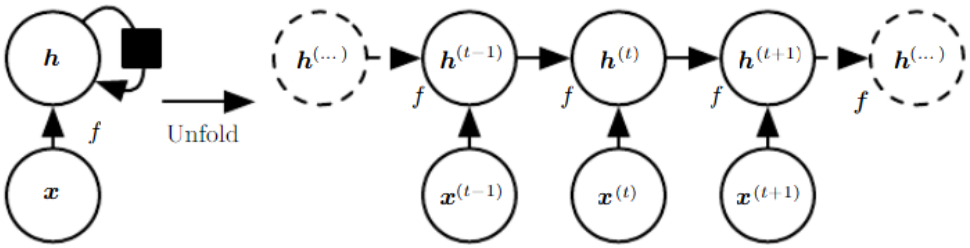

Figura 3.18. Rappresentazione di una rete neurale ricorrente in forma "ripiegata" e "dispiegata" [19, p. 370].

La scelta della funzione di attivazione τ da utilizzare per gli strati nascosti della rete ricorrente può ricadere su ciascuna delle tipologie già analizzate finora, a seconda della struttura dei dati e del problema in esame; in alternativa, risulta ampiamente utilizzata nella pratica anche un ulteriore funzione, nota come "tangente iperbolica" [19, p. 374], o più comunemente detta tanh (pronunciato "tanch") [23, cap.3].
La funzione di attivazione "tanh" rappresenta l'output di uno strato nascosto nella forma seguente [23, cap.3]:

$$\text{Output} = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{3.14}$$

Dove il termine z è già stato definito nella sezione 3.4 come la somma pesata degli input in quel determinato strato della rete [23, cap.3].

A differenza di molte funzioni viste finora, la tangente iperbolica assume qualunque valore reale compreso tra -1 e 1 [23, cap.3], rendendone l'utilizzo piuttosto comune ai livelli intermedi di una rete, rispetto ad altre forme, come la funzione sigmoidea e il softmax, utilizzate quasi esclusivamente negli strati finali.

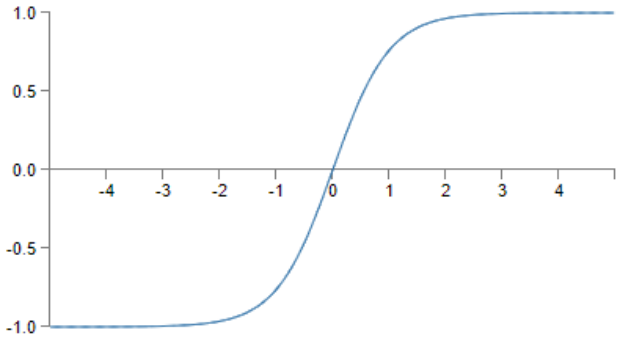

Figura 3.19. Funzione di attivazione tangente iperbolica (tanh) [23, cap.3]

Capitolo 4

Metodologie per l'addestramento di reti neurali

Il precedente capitolo ha fornito un'esaustiva introduzione alla trattazione delle reti neurali artificiali, illustrandone le caratteristiche essenziali e mettendone in luce le tipologie principali, che hanno trovato larga applicazione nelle ricerche empiriche condotte negli anni su questa specifica area di interesse.
Nel dettaglio, si è visto come alla base di ciascuna architettura di rete neurale vi siano principalmente due parametri: i "pesi", $\omega_1, \omega_2, ..., \omega_n$, e il bias, b; tuttavia, seppur sia stata offerta una panoramica ampia sul significato di ciascuno di questi valori, nessuna indicazione specifica è stata data sui criteri che stanno alla base della loro "scelta", e, di conseguenza, sulle tecniche di addestramento utilizzabili per costruire degli efficaci modelli appartenenti a questa complessa famiglia di algoritmi.
I prossimi paragrafi mirano a "coprire" gli spazi lasciati in tal senso dalle sezioni precedenti, illustrando le modalità attraverso cui sono selezionati i parametri di un generico modello di rete neurale, e discutendo dettagliatamente le principali tecniche algoritmiche (discesa del gradiente e backpropagation), che ne consentono la costruzione e ne

determinano la fase di "training"; infine, nella parte conclusiva del capitolo, l'attenzione sarà focalizzata sui modelli di natura "feed-back" e sui metodi fondamentali che permettono l'addestramento di una rete neurale ricorrente.

4.1 Funzioni di costo e scelta dei parametri del modello

Nell'introduzione al presente capitolo si è mostrato come l'efficacia di un modello di rete neurale dipenda, essenzialmente, dalla scelta dei suoi parametri costitutivi: l'insieme dei pesi $\omega_1, \omega_2, ..., \omega_n$, e il bias b [23, cap.1]. Questo perchè, come si è visto già nel paragrafo 3.2, qualunque tipologia di attivazione presente all'interno di uno strato della rete, a prescindere dalla forma della funzione ad essa associata, e per ogni genere di architettura del modello, può essere rappresentata come [23, cap.1]:

$$a = f(x_n \cdot \omega_n + b) \tag{4.1}$$

Da tale espressione, infatti, risulta evidente il ruolo svolto dai pesi ω_n e dal bias b ai fini della determinazione dell'output finale del modello, che non costituisce altro che la predizione (sia essa espressa in forma numerica o in termini di "classe") necessaria alla risoluzione di uno specifico problema reale [23, cap.1].
In termini generici, essendo per l'appunto la predizione di valori numerici o categorici il fine ultimo di qualunque algoritmo di apprendimento automatico, anche l'efficacia di una rete neurale può essere determinata sulla base della rispettiva capacità di ottenere dei risultati finali che siano quanto più "prossimi" possibile ai dati presenti nel set di "testing" [23, cap.1].
In tal senso, è possibile definire una "funzione di costo" che rappresenti la "perdita", in termini di efficacia predittiva, determinata dal modello rispetto ai valori reali che, in una fase successiva, sono ad esso rapportati [23, cap.1]:

$$C = f(\omega, b) \tag{4.2}$$

Questa funzione è, come precedentemente anticipato, dipendente da due parametri fondamentali, ossia i vettori dei pesi e dei bias della rete [23, cap.1]; inoltre, a prescindere dalla forma specifica che essa può andare ad assumere, ogni funzione di costo tende a 0 quanto più l'output del modello $\hat{y}$ si avvicina ai valori reali di "testing" y [23, cap.1]. Ciò significa, in estrema sostanza, che, al fine di ottenere delle predizioni accurate ed efficienti, sarà necessario determinare una combinazione di ω e b tale da minimizzare la funzione di costo utilizzata per valutare il modello [23, cap.1]; la scelta della funzione da utilizzare, naturalmente, è determinante ai fini dell'individuazione dei parametri "ottimali" del modello, e, per tale ragione, deve essere effettuata in assoluta armonia con gli scopi e le modalità delle ricerche per cui l'architettura di rete neurale è stata realizzata [23, cap.1].

Una prima tipologia di funzione di costo, dalla struttura indubbiamente semplice e che ha goduto di ampio utilizzo in letteratura, è il c.d. "errore quadratico medio", o "mean-squared error" (MSE), la cui espressione algebrica è la seguente [23, cap.1]:

$$C(\omega, b) = \frac{1}{2n} \cdot \sum_{x} [\hat{y}(x) - y]^2 \tag{4.3}$$

Dove con x si intendono gli input del modello di rete neurale, con n la numerosità delle osservazioni iniziali e, rispettivamente, con $\hat{y}$ e y i valori effettivi della variabile dipendente e gli output generati dall'algoritmo, per ciascun input x [23, cap.1].

Una delle caratteristiche principali del MSE è la non-negatività: infatti, ciascuno degli elementi della sommatoria presente nella (4.3) può assumere unicamente valori ≥ 0 [23, cap.1]; inoltre, come ogni altra funzione di costo, vale la condizione per cui $C(\omega, b) \approx 0$ se $\hat{y} \approx y$, e, di conseguenza, le predizioni generate dal modello di rete neurale risulteranno tanto più accurate quanto maggiore sarà la prossimità del MSE a 0 [23, cap.1].

Il principale punto di forza di questa funzione, che ne ha determinato l'ampio utilizzo nelle applicazioni empiriche, è la sua notevole semplicità strutturale [23, cap.1]: fondamentalmente, il MSE non è altro che una media "aggiustata" della somma degli scarti tra i valori reali del dataset di "testing" utilizzato per validare la rete, osservabili ex post, e i valori predetti dal modello; il risultato finale di questa operazione è infine

elevato al quadrato per evitare che la funzione possa restituire come proprio output un numero negativo.

Di contro, uno dei principali problemi connessi all'utilizzo del MSE, per un modello di rete neurale, è la forte sensibilità di questa funzione di costo alla presenza di possibili osservazioni "outlier"[1] all'interno del dataset utilizzato per l'addestramento della stessa [37, p. 1249]; tale comportamento è essenzialmente dovuto, come facilmente intuibile, alla presenza di un elevamento a potenza all'interno dell'espressione riportata nella (4.3), che fa sì che valori "anomali", in eccesso o in difetto, rispetto alla media delle osservazioni, acquisiscano un peso notevole nel calcolo del valore finale della perdita associata al modello [37, p. 1249].

Per far fronte a tale imprevisto, che può senza dubbio rappresentare una fonte di criticità, sono state elaborate ulteriori funzioni di costo dall'apparenza estremamente simile al MSE, che tuttavia mirano a eliminarne i difetti principali, come il c.d. "errore assoluto medio", o "mean-absolute error" (MAE), che descrive la perdita connessa ad un particolare modello come segue [19, p. 181]:

$$C(\omega, b) = \frac{1}{n} \cdot \sum_{x} |\hat{y}(x) - y| \tag{4.4}$$

In questo caso, come è facilmente osservabile dall'equazione, il "costo" generato da un modello di rete neurale è rappresentato come media aritmetica degli scarti tra i valori osservabili nel dataset e le predizioni fornite dall'algoritmo, di cui successivamente è computato il valore assoluto per evitare l'insorgenza di possibili risultati negativi [19, p. 181]; in questo modo, è possibile far fronte alla scarsa efficacia predittiva in presenza di dati "outlier", presentata dalla funzione MSE, senza tuttavia dover elevare al quadrato l'intero termine della sommatoria nella (4.4) [19, p. 181].

Anche il MAE, tuttavia, presenta un ragguardevole svantaggio strutturale, se comparato al mean-squared error: l'utilizzo del valore assoluto, infatti, pur non determinando

[1] In statistica, si definiscono "outlier" dei valori "anomali" e decisamente distanti (in eccesso o in difetto) dal complesso delle osservazioni presenti nel dataset, al punto da poter essere generati da errori di rilevazione o di misura.

problemi particolari in caso in cui il modello sia addestrato utilizzando valori "anomali", genera non pochi problemi in presenza di calcoli matematici complessi[2], che rendono il ricorso a quest'espressione algebrica, nelle specifiche applicazioni ivi proposte, particolarmente indesiderabile [37, p. 1249].

Inoltre, entrambe le funzioni di costo analizzate in precedenza, hanno presentato, in diverse applicazioni pratiche recenti, una notevole difficoltà nelle fasi iniziali dell'addestramento di reti neurali, che, in presenza di un loro utilizzo, risulta avvenire in tempi decisamente più lunghi della media (c.d. "learning slowdown") [23, cap.3].

Anche in virtù di questa ulteriore complicanza, non è raro rinvenire nelle più moderne applicazioni empiriche il ricorso ad una tipologia di funzione di costo completamente differente dalle due, decisamente simili, osservate finora, definita anche "entropia incrociata" o, nella sua accezione originale, "cross-entropy" [23, cap.3]; tale funzione, basata sull'utilizzo dei logaritmi, può essere rappresentata algebricamente come segue [23, cap.3]:

$$C(\omega, b) = \frac{1}{n} \cdot \sum_{x} [\hat{y}(x) \cdot \ln y + (1 - \hat{y}(x)) \cdot \ln(1 - y)] \tag{4.5}$$

Anche la funzione di "cross-entropy", come già osservato per il MSE, è una funzione non negativa, che tende a 0 quanto più i valori osservati sono prossimi a quelli "predetti" dalla rete neurale [23, cap.3]; inoltre, la quantità $[\hat{y}(x) \cdot \ln y + (1 - \hat{y}(x)) \cdot \ln(1 - y)]$, utilizzata come termine della sommatoria, è spesso definita nella letteratura statistica con il nome di "funzione di entropia binaria", in quanto utilizzata per descrivere la distribuzione dell'entropia di un generico processo probabilistico bernoulliano[3] [23, cap.3].

Le tre funzioni di costo esposte in questo paragrafo, ossia il "mean-squared error", il "mean-absolute error" e la "cross-entropy", pur rappresentando senza dubbio le misure più comunemente utilizzate per quantificare la perdita generata da un modello di rete

[2]Che, come vedremo nelle prossime sezioni, sono necessari per determinare il minimo assoluto della funzione di costo e calcolarne il gradiente.

[3]In statistica, si intende con la locuzione "processo bernoulliano" una sequenza di variabili casuali binarie, che assumono unicamente i valori 0 e 1.

neurale, non esauriscono certo la vasta gamma di soluzioni adottabili a tale scopo; tuttavia, per naturali esigenze di sintesi dovute alla natura della trattazione in essere, i successivi paragrafi si concentreranno piuttosto sull'analisi delle principali tecniche connesse alla minimizzazione di una generica funzione di costo, allo scopo di individuare le modalità attraverso cui possa avvenire una selezione ottimale dei parametri costituenti una rete neurale [23, cap.1].
Più dettagliatamente, il successivo paragrafo illustrerà la principale famiglia di tecniche algoritmiche ideate a tale scopo, basate sul concetto di "gradiente" di una funzione di costo [23, cap.1]; la sezione 4.3, infine, mostrerà in che modo tale gradiente può essere calcolato attraverso un procedimento noto in letteratura con il nome di "backpropagation" [23, cap.2].

4.2 Algoritmi di minimizzazione della funzione di costo: discesa del gradiente e SGD

Nella precedente sezione, è stato introdotto il fondamentale concetto di "funzione di costo", che, essenzialmente, va a designare una quantificazione della perdita generata da un determinato modello di apprendimento automatico, nel caso specifico in esame, le reti neurali artificiali [23, cap.1].
Successivamente, si è asserito che un criterio generalmente accettato di scelta dei parametri fondanti di un rete neurale, ossia i pesi e il bias, è, per l'appunto, l'utilizzo dei valori che generinino una minimizzazione della funzione di costo, garantendo, in tal modo, che il modello presenti la minor perdita possibile e, di conseguenza, possa effettuare delle predizioni più accurate e "vicine" ai dati utilizzati per validarlo [23, cap.1].
Ma in che modo può essere calcolato il minimo di una generica funzione di costo?
Si supponga di avere una funzione C, dipendente da due parametri, ossia un vettore

dei pesi della rete, ω, e un vettore dei bias, b [23, cap.1]:

$$C = f(\omega, b) \tag{4.6}$$

Questa funzione di costo, definita in R^3, può essere rappresentata come una superficie su uno spazio tridimensionale, come mostra la figura successiva [23, cap.1]:

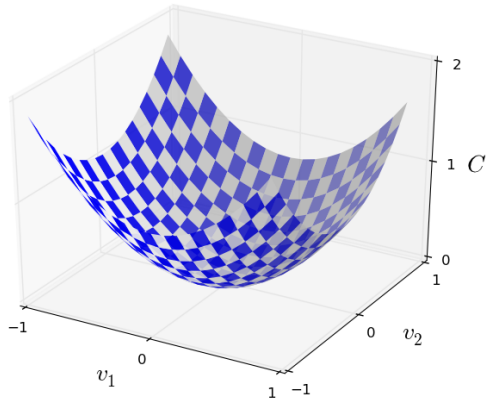

Figura 4.1. Struttura di una generica funzione di costo a due variabili [23, cap.1].

Come si può osservare, la funzione in figura presenta un minimo assoluto, in corrispondenza del punto in cui si crea l'inarcamento a forma di "valle" [23, cap.1]; l'obiettivo dell' algoritmo di minimizzazione sarà, in base a quanto detto in precedenza, "discendere" la funzione di costo, partendo da un iniziale punto $C_1 = (\omega_1, b_1)$, determinato in maniera casuale, per giungere infine al suo minimo assoluto, cui corrisponderà una combinazione di parametri $(\omega*, b*)$ ottimale [23, cap.1].

Per "discendere" la funzione di costo, dal punto di partenza identificato, arrivando al minimo globale della stessa, è necessario far "variare" i parametri del modello di un dato ammontare, che in questo caso si può definire, rispettivamente, $\Delta\omega$ per il vettore dei pesi e Δb per quello dei bias [23, cap.1]; conseguenzialmente, per una nota regola del calcolo differenziale, la variazione complessiva della funzione di costo associata alle

variazioni dei due parametri sarà approssimativamente pari a [23, cap.1]:

$$\Delta C \approx \frac{\partial C}{\partial \omega} \cdot \Delta \omega + \frac{\partial C}{\partial b} \cdot \Delta b \tag{4.7}$$

Dove, rispettivamente, $\frac{\partial C}{\partial \omega}$ e $\frac{\partial C}{\partial b}$ sono le derivate parziali della funzione di costo rispetto al parametro ω e al bias b [23, cap.1].
Convenzionalmente, l'espressione (4.7) può essere reinterpretata ricorrendo al concetto di "gradiente" ∇C di una funzione a più variabili, che può essere definito come il vettore delle sue derivate parziali $(\frac{\partial C}{\partial \omega}, \frac{\partial C}{\partial b})$ [23, cap.1]; seguendo questa notazione, è possibile riscrivere l'equazione precedente come segue [23, cap.1]:

$$\Delta C \approx \nabla C \cdot \Delta p \tag{4.8}$$

Dove, per l'appunto, ∇C rappresenta il gradiente della funzione di costo e Δp la variazione complessiva dei due parametri del modello, ω e b [23, cap.1].
Il gradiente ∇C determina che variazioni nei pesi e nei bias si riflettano in variazioni complessive della funzione di costo C [23, cap.1]; ciò significa, in termini maggiormente esplicativi, che anche la stessa oscillazione dei parametri Δp può essere espressa come funzione del vettore delle derivate parziali di C, come segue:

$$\Delta p = \eta \cdot \nabla C \tag{4.9}$$

Dove η è un valore che determina l'entità della variazione dei pesi, o, per utilizzare un'espressione più chiara, la "forza" con cui ci si muove all'interno del gradiente della funzione di costo [23, cap.1]; in questo senso, l'intera equazione originaria (4.7) può essere riscritta come [23, cap.1]:

$$\Delta C \approx \nabla C \cdot (\eta \cdot \nabla C) \tag{4.10}$$

Per cui risulta evidente che le variazioni complessive a livello della funzione di costo siano direttamente proporzionali all'andatura del gradiente (che, del resto, non è altro

che una rappresentazione sintetica del modo in cui tale funzione varia in rapporto ai suoi parametri costitutivi) [23, cap.1]; a tal proposito, utilizzando valori negativi del parametro η, è possibile "discendere" il gradiente della funzione di costo, allo scopo di raggiungere il punto corrispondente al suo minimo globale [23, cap.1].
Proprio in virtù di questa caratteristica fondante, la tecnica di minimizzazione descritta sinora è definita anche "discesa del gradiente", o "gradient descent", e costituisce l'algoritmo maggiormente utilizzato per la risoluzione di problemi connessi alla scelta ottimale dei parametri di un modello di rete neurale artificiale [23, cap.1].
Inoltre, il parametro η, rappresentando la "velocità" con cui è disceso il gradiente alla ricerca del minimo assoluto di C, è anche detto "tasso d'apprendimento" o "learning rate", in quanto esprime, per l'appunto, l'intensità con cui avviene l'addestramento del modello: a valori molto alti di η, infatti, corrisponde una maggiore rapidità nella discesa della funzione di costo, dando luogo alla possibilità concreta di terminare la fase di addestramento della rete in presenza di punti sub-ottimali [23, cap.1]; di contro, tuttavia, a valori molto bassi del learning rate corrisponde una discesa decisamente più "lenta", che può determinare l'insorgenza di un "training" particolarmente lungo per il modello, pur comunque generando dei risultati finali tendenzialmente preferibili [23, cap.1].

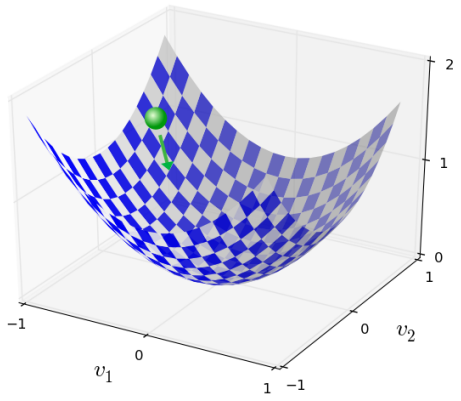

Figura 4.2. La discesa del gradiente di una funzione di costo [23, cap.1].

La figura in alto rappresenta sinteticamente le modalità attraverso cui si verifica la discesa del gradiente di una funzione di costo: il "pallino" verde, rappresentante il vettore iniziale dei pesi e dei bias, costituisce il punto di partenza per l'algoritmo, che successivamente "aggiusta" tali valori in modo tale da raggiungere, con un'intensità data dal learning rate η, il minimo assoluto della funzione di costo.
Una notevole complicazione nel processo di computazione del gradiente sopraggiunge nei casi in cui la funzione di costo possa essere "scritta" come media o somma di più funzioni differenziabili (cosa che, nella pratica, avviene non di rado, se si pensa anche alle tre funzioni analizzate nel precedente paragrafo che sono tutte in possesso di tale caratteristica) [23, cap.1]; si pensi, ad esempio, a una funzione di costo che abbia la forma seguente [23, cap.1]:

$$C(\omega, b) = \frac{1}{n} \sum_{x} C_x \tag{4.11}$$

Dove con x si intende il numero di input della rete neurale e con n il numero di osservazioni utilizzate per addestrarla [23, cap.1]; inoltre, come si può facilmente osservare, la funzione di costo non è altro che la media aritmetica delle x funzioni di costo per ciascun input del modello [23, cap.1].
In questo caso, utilizzando il procedimento illustrato in precedenza per il calcolo del gradiente, sarebbe necessario calcolare ∇C effettuando la media aritmetica dei singoli gradienti ∇C_x ricavati da ciascuno degli x input del modello, come segue [23, cap.1]:

$$\nabla C = \frac{1}{n} \sum_{x} \nabla C_x \tag{4.12}$$

Risulta evidente che, all'aumentare degli input della rete, anche la complessità dell'espressione (4.12) incrementerebbe notevolmente, in quanto sarebbe necessario sommare un numero elevatissimo di gradienti per computare quello "generale" della funzione di costo da minimizzare [23, cap.1]; per tale ragione, in casi come quello in esame, è spesso utilizzata una variante dell'algoritmo standard di discesa del gradiente, nota come "discesa stocastica del gradiente", o "SGD" (dal corrispettivo termine inglese, "stochastic gradient descent"), la quale permette di stimare il valore ∇C utilizzando soltanto

un piccolo campione, pari a $m < n$, di input, definito anche "mini-batch" (mentre il parametro m è comunemente definito "dimensione del campione", o "batch size") [23, cap.1].

L'idea alla base della SGD è che, scegliendo una dimensione m del mini-batch sufficientemente elevata, il valore medio dei gradienti ∇C_{xj}, computati sugli input presenti nel campione, tenderà alla media complessiva dei gradienti di tutti gli input, ∇C_x [23, cap.1]:

$$\frac{1}{m}\sum_{xj}\nabla C_{xj} \approx \frac{1}{n}\sum_{x}\nabla C_x = \nabla C \tag{4.13}$$

Utilizzando l'algoritmo SGD, l'addestramento della rete è effettuato, in estrema sintesi, su gruppi di "mini-batch", piuttosto che su un unico insieme comprendente tutti gli input del modello [23, cap.1]. Nello specifico, una volta che l'intero numero n delle osservazioni originarie è stato utilizzato per completare una fase di "training", si usa dire che la rete ha compiuto un'intero ciclo, o "epoca" (epoch), di addestramento [23, cap.1]; una volta terminata la prima epoca di "training" del modello, ne viene iniziata una successiva, e così via, per un numero complessivo di volte determinato sulla base delle caratteristiche della ricerca in esame [23, cap.1].

Il corretto calcolo del gradiente, dunque, per qualunque input del modello e in ciascuno strato della rete, è un presupposto necessario per implementare i suddetti algoritmi di minimizzazione, fondamentali per la "scelta" dei parametri ottimali da utilizzare per la costruzione di una rete neurale; tuttavia, in questo paragrafo, nessuna indicazione specifica è stata fornita sugli strumenti metodologici utilizzati per computare ∇C.

La prossima sezione mirerà a completare la trattazione iniziata in questa sede, illustrando una metodologia particolarmente utilizzata in letteratura per il calcolo del gradiente di una funzione di costo, nota anche come "algoritmo di backpropagation" [23, cap.2].

4.3 Un algoritmo per la computazione del gradiente: la backpropagation

La sezione appena conclusa ha fornito una panoramica esaustiva sulle principali tecniche di minimizzazione di una funzione di costo, necessarie per l'addestramento di una rete neurale artificiale, e basate sul concetto di gradiente ∇C di tale funzione, inteso come vettore delle sue derivate parziali rispetto ai pesi, ω, e ai bias, b [23, cap.1]; questo paragrafo mira ad offrire al lettore una descrizione rigorosa di una metodologia algoritmica largamente utilizzata in letteratura ai fini della computazione del gradiente di C, nota anche con il nome di "backpropagation" [23, cap.2].

L'algoritmo di backpropagation (forma contratta dell'espressione "backward propagation of errors", lett. "propagazione all'indietro degli errori") fu introdotto negli anni settanta [23, cap.2], pur tuttavia trovando notevole consenso in dottrina solamente a partire dal 1986, anno di pubblicazione dell'articolo "Learning representations by back-propagating errors", da parte di David E. Rumelhart, Geoffrey E. Hinton e Ronald J. Williams [23, cap.2] [7]; questo lavoro, infatti, evidenziò l'efficienza di tale metodo di addestramento di reti neurali, se comparato ad altri algoritmi maggiormente utilizzati all'epoca ma decisamente più "lenti" e meno accurati [23, cap.2].

Questa tecnica si basa, come il nome ad essa attribuito suggerisce, sul concetto di "errore", inteso come saggio di variazione della funzione di costo, per uno specifico neurone j in un dato strato l della rete, rispetto alle oscillazioni della somma pesata degli input connesi a j nel medesimo strato, z_j^l [23, cap.2]:

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} \tag{4.14}$$

L'interpretazione di δ_j^l, seppur possa apparire a prima vista complessa, è in realtà estremamente intuitiva: questo termine, infatti, non è altro che la quantificazione degli "scostamenti" nella funzione generale di costo attribuibili a variazioni nella somma pesata degli input del j-esimo neurone all'interno del l-esimo strato, rappresentando, per tale ragione, la porzione della perdita complessivamente generata dal modello ad

esso attribuibile (e dunque, in estrema sintesi, il suo "errore") [23, cap.2].
L'algoritmo di backpropagation, come si avrà modo di osservare, permette di computare il termine di errore δ_j^l per ciascun insieme di neuroni e strati di cui si compone la rete, utilizzando tale valore come base per il calcolo delle quantità di interesse ($\frac{\partial C}{\partial \omega}$ e $\frac{\partial C}{\partial b}$) che descrivono il gradiente ∇C della funzione di costo [23, cap.2].
Più dettagliatamente, il primo passo da compiere per implementare questo metodo di addestramento, è il calcolo dell'errore δ_j^L associato a ciascuno dei j neuroni che compongono lo strato finale di output L, attraverso la formula che segue [23, cap.2]:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \cdot \sigma(z_j^L) \tag{4.15}$$

Dove, nello specifico, a_j^L è la j-esima attivazione presente nello strato di output e $\sigma(z_j^L)$ la rispettiva funzione di attivazione (necessariamente continua e differenziabile) utilizzata per quel livello della rete [23, cap.2]; l'intera espressione, che costituisce anche il c.d. "forward pass" dell'algoritmo di backpropagation [23, cap.2], è direttamente ricavata da una nota formula per il calcolo delle derivate di funzioni composte, nota anche come "regola della catena"[4] [23, cap.2].
Ma in che modo la regola della catena è applicata per ottenere l'eguaglianza osservata nella (4.15)?
Innanzitutto, si consideri che, come visto in precedenza nell'espressione (4.14), l'errore δ_j^L è per definizione pari a [23, cap.2]:

$$\delta_j^L = \frac{\partial C}{\partial z_j^L} \tag{4.16}$$

[4]Nota regola del calcolo infinitesimale secondo cui, date due funzioni continue e differenziabili, $y = f(z)$ e $z = g(x)$, la derivata di y rispetto ad x è esattamente uguale a $\frac{dy}{dx} = \frac{dy}{dz} \cdot \frac{dz}{dx}$ [19, p. 205].

Applicando la formula per il calcolo delle derivate di funzioni composte, è possibile reinterpretare la (4.16) in relazione all'attivazione a_j^L, come segue [23, cap.2]:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \cdot \frac{\partial a_j^L}{\partial z_j^L} \tag{4.17}$$

Di conseguenza, essendo l'attivazione $a_j^L = \sigma(z_j^L)$, l'ultima equazione sarà esattamente equivalente alla (4.15), per cui $\delta_j^L = \frac{\partial C}{\partial a_j^L} \cdot \sigma(z_j^L)$ [23, cap.2].
Una volta computato l'errore δ_j^L relativo al j-esimo neurone dello strato finale della rete, gli "step" successivi compiuti dall'algoritmo di backpropagation permettono di utilizzare quest'ultimo per calcolare ogni singolo termine di errore relativo a ciascun k-esimo neurone presente nei livelli inferiori del modello [23, cap.2]; infatti, per ragioni che saranno discusse a breve, il valore δ_k^{L-1} relativo all'errore di un generico neurone k presente nello strato precedente rispetto a quello di output, può essere espresso in funzione di δ_j^L, come segue [23, cap.2]:

$$\delta_k^{L-1} = \sum_j \omega_{jk}^L \cdot \delta_j^L \cdot \sigma(z_k^{L-1}) \tag{4.18}$$

L'espressione (4.18) può, inoltre, essere reinterpretata in una forma maggiormente generica, nel modo seguente [23, cap.2]:

$$\delta_j^l = \sum_k \omega_{kj}^{l+1} \cdot \delta_k^{l+1} \cdot \sigma(z_j^l) \tag{4.19}$$

Dove δ_j^l rappresenta il valore dell'errore relativo al j-esimo neurone dello strato l, ed è del tutto speculare al termine presente nella (4.14) [23, cap.2]; δ_k^{l+1}, di contro, rappresenta l'errore relativo al k-esimo neurone dello strato immediatamente successivo, $l+1$, mentre ω_{kj}^{l+1} rappresenta il "peso" della connessione tra il j-esimo neurone dello strato l e il k-esimo neurone dello strato $l+1$ [23, cap.2]. Infine, il valore $\sigma(z_j^l)$ rappresenta, come già osservato in precedenza, la funzione di attivazione relativa al j-esimo neurone all'interno dello strato l della rete neurale [23, cap.2].

Anche quest'ultima, fondamentale espressione, deriva da un'applicazione della già citata regola della catena [23, cap.2]; infatti, come visto nell'equazione (4.14), il termine δ_j^l può essere riscritto come $\frac{\partial C}{\partial z_j^l}$, il quale, a sua volta, può essere ulteriormente interpretato in relazione al k-esimo errore nello strato successivo della rete, $\delta_k^{l+1} = \frac{\partial C}{\partial z_k^{l+1}}$, come segue [23, cap.2]:

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} = \sum_k \frac{\partial C}{\partial z_k^{l+1}} \cdot \frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l} \cdot \delta_k^{l+1} \tag{4.20}$$

Inoltre, essendo il termine z_k^{l+1} equivalente alla somma pesata degli input di quello specifico neurone nello strato $l+1$, si può ottenere la seguente uguaglianza [23, cap.2]:

$$z_k^{l+1} = \sum_j \omega_{kj}^{l+1} \cdot a_j^l + b_k^{l+1} = \sum_j \omega_{kj}^{l+1} \cdot \sigma(z_j^l) + b_k^{l+1} \tag{4.21}$$

Dall'espressione (4.21) appena osservata, è possibile ricavare la condizione per cui [23, cap.2]:

$$\frac{\partial z_k^{l+1}}{\partial z_j^l} = \omega_{kj}^{l+1} \cdot \sigma(z_j^l) \tag{4.22}$$

La quale, sostituita nella (4.20), consente di ottenere esattamente l'equazione iniziale, ossia $\delta_j^l = \sum_k \omega_{kj}^{l+1} \cdot \delta_k^{l+1} \cdot \sigma(z_j^l)$ [23, cap.2].

L'algoritmo di backpropagation consente, una volta compiuto il "forward pass", calcolando il termine di errore per lo strato finale della rete, di "propagare" tale errore negli strati inferiori, giungendo infine a computare il gradiente complessivo della funzione di costo ∇C (il c.d. "backward pass") [23, cap.2].

Più specificamente, utilizzando ancora una volta la regola della catena, è possibile dimostrare che la derivata parziale della funzione di costo rispetto ad un qualunque peso presente all'interno della rete è esattamente pari a [23, cap.2]:

$$\frac{\partial C}{\partial \omega_{jk}^l} = a_k^{l-1} \cdot \delta_j^l \tag{4.23}$$

Simmetricamente, la derivata parziale di C rispetto al j-esimo bias all'interno dello strato l della rete neurale, è calcolabile attraverso la seguente espressione [23, cap.2]:

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{4.24}$$

Applicando le ultime due equazioni sopra descritte, è dunque possibile, in estrema sintesi, computare il gradiente della funzione di costo ∇C, calcolando ciascuna derivata parziale rispetto ai pesi e ai bias, e utilizzando a tal proposito, per ogni strato della rete, l'errore δ_j^l, calcolabile attraverso il meccanismo della "propagazione" dal livello finale di output a quelli intermedi, per giungere infine agli strati iniziali [23, cap.2].
L'algoritmo di backpropagation descritto in questo paragrafo può essere utilizzato, in combinazione con una delle tecniche di minimizzazione basate sul gradiente osservate in precedenza, per la scelta dei parametri ottimali di una qualunque architettura di rete neurale "feed-forward" [23, cap.2]; nella prossima sezione, che concluderà la trattazione legata al presente capitolo, verrà illustrata una metodologia algoritmica designata allo scopo di "addestrare" reti neurali di tipologia "feed-back", nota anche come "backpropagation through time" [19, p. 384].

4.4 Addestrare reti neurali "feed-back" attraverso la backpropagation through time

Il paragrafo precedente ha illustrato le principali caratteristiche dell'algoritmo di backpropagation, descrivendone nel dettaglio il funzionamento ed illustrandone le espressioni matematiche che ne stanno alla base [23, cap.2]; nello specifico, si è osservato come tale metodologia possa essere utilizzata per addestrare reti neurali di tipo "feed-forward", laddove per le reti ricorrenti venga preferita una variante della stessa, nota anche con il nome di "backpropagation through time" [19, p. 384].
In realtà, a dispetto del suo nome indubbiamente suggestivo, l'algoritmo di backpropagation through time non presenta differenze sostanziali rispetto alla versione "standard"

osservata nella sezione 4.3, rappresentandone, più che altro, un'applicazione specifica alle reti neurali ricorrenti [19, pp. 384-385].
Più dettagliatamente, come si è visto nella sezione 3.5, una rete neurale ricorrente può essere rappresentata in una forma "ripiegata", che ne descriva le caratteristiche fondamentali in una forma maggiormente "implicita" [19, p. 370]:

$$h^{(t)} = f(h^{(t-1)}, x^{(t)}, \tau) \tag{4.25}$$

Oppure, con un significato ed una struttura assolutamente analoghi alla precedente, la (4.25) può essere riscritta nella forma di un'espressione più "esplicita", detta anche rete ricorrente "dispiegata" [19, p. 370]:

$$h^{(t)} = f(f(h^{(t-2)}; x^{(t-1)}; \tau), x^{(t)}, \tau) \tag{4.26}$$

L'algoritmo di backpropagation through time consiste, essenzialmente, nell'utilizzo delle medesime formule osservate per la versione "base" del metodo di backpropagation, ai fini di propagare l'errore δ_j^l, dallo strato finale di output della rete ricorrente, agli strati iniziali, computandone in tal modo il gradiente complessivo ∇C [19, pp. 384-385]; l'insieme di quete operazioni, come si può facilmente intuire, è effettuato su una rete ricorrente "dispiegata", come quella osservata nella (4.26), la cui rappresentazione maggiormente "esplicita" ne consente una più agevole sottoposizione alle metodologie algoritmiche descritte in precedenza [19, pp. 384-385].
Tuttavia, un problema di notevole rilevanza connesso all'utilizzo di algoritmi di addestramento basati sulla discesa del gradiente di una funzione C, è legato alla possibile "instabilità" dello stesso, laddove la rete sia particolarmente "profonda", con un elevato numero di strati nascosti [23, cap.5]; inoltre, tale comportamento si è osservato con ampia frequenza nelle architetture neurali di tipo "feed-back", rendendo le reti ricorrenti uno dei modelli principalmente soggetti a questa forma di complicazione [32].
Ma che cosa si intende, per l'esattezza, con "instabilità" del gradiente di una funzione di costo [23, cap.5]?

Per comprendere pienamente tale concetto, si faccia riferimento al precedente paragrafo: applicando la regola della catena, si è osservato come la derivata parziale della funzione di costo rispetto ad un qualunque peso presente nella rete, potesse essere computabile come [23, cap.2]:

$$\frac{\partial C}{\partial \omega_{jk}^{l}} = a_{k}^{l-1} \cdot \delta_{j}^{l} \tag{4.27}$$

Simmetricamente, anche lo scostamento di C, in rapporto ad un generico bias presente in un dato strato della rete, può essere calcolato attraverso l'algoritmo di backpropagation, come segue [23, cap.2]:

$$\frac{\partial C}{\partial b_{j}^{l}} = \delta_{j}^{l} \tag{4.28}$$

L'equazione (4.28), relativa al calcolo di ∇C con riferimento al bias b_{j}^{l}, può essere ulteriormente riscritta, facendo esplicito riferimento a ciascuna delle sue componenti costitutive[5] [23, cap.5]:

$$\frac{\partial C}{\partial b_{j}^{l}} = \delta_{j}^{l} = \sum_{k} \omega_{kj}^{l+1} \cdot \delta_{k}^{l+1} \cdot \sigma(z_{j}^{l}) \tag{4.29}$$

Analogamente, anche l'equazione (4.27), relativa al calcolo della variazione di C rispetto a un generico peso ω_{jk}^{l}, può subire una "rivisitazione" del tutto speculare a quanto osservato nell'espressione precedente [23, cap.5]:

$$\frac{\partial C}{\partial \omega_{jk}^{l}} = a_{k}^{l-1} \cdot \delta_{j}^{l} = a_{k}^{l-1} \cdot \sum_{k} \omega_{kj}^{l+1} \cdot \delta_{k}^{l+1} \cdot \sigma(z_{j}^{l}) \tag{4.30}$$

Come si può chiaramente osservare, entrambe le equazioni rappresentanti il gradiente ∇C, dipendono dal termine di sommatoria $\sum_{k} \omega_{kj}^{l+1}$ [23, cap.5].
Ciò significa che, per valori significativamente piccoli dei pesi, ad esempio tali che

[5] Per maggiori chiarimenti riguardo questa affermazione, si osservi l'equazione (4.19) all'interno della sezione precedente.

$|\omega_{kj}^{l+1}| < 1$, il gradiente della funzione tenderà a "diminuire" progressivamente all'avvicinarsi della propagazione di δ_j^l verso gli strati iniziali della rete [23, cap.5]; per tale singolare comportamento di ∇C in presenza delle condizioni sovracitate, il suddetto fenomeno è detto anche "scomparsa del gradiente" (nell'accezione originaria, "vanishing gradient") [23, cap.5].

Di contro, nel caso opposto in cui i pesi ω_{kj}^{l+1} assumano valori significativamente elevati, in modo tale che $|\omega_{kj}^{l+1}| > 1$, il gradiente di C tenderà ad aumentare progressivamente al procedere del "backward pass" verso lo strato iniziale della rete [23, cap.5]; in questo caso specifico, il comportamento di ∇C dà origine a un fenomeno noto in letteratura come "esplosione del gradiente" (o, utilizzando l'espressione originaria, "exploiding gradient") [23, cap.5].

L'instabilità del gradiente, che può essere dovuta ad una qualunque delle due forme osservate in precedenza, rende sostanzialmente problematico l'addestramento di reti neurali con diversi strati intermedi, in quanto sia l'algoritmo di backpropagation, che la sua variante applicata ai modelli "feed-back", necessiterebbero di un numero di "step" maggiori per completare il "backward pass", determinando dunque la possibile sparizione o esplosione di ∇C [23, cap.5]; inoltre, come già anticipato in precedenza, tale fenomeno è riscontrato maggiormente nelle reti neurali ricorrenti, generando un'incapacità da parte di tali modelli nell'ottenere predizioni soddisfacenti laddove essi siano usati per rappresentare sequenze di dati temporalmente "distanti" gli uni dagli altri (in tal senso, si suol dire che reti neurali di questo genere siano incapaci di cogliere le c.d. "dipendenze di lungo periodo") [32].

Per ovviare a tale criticità, le reti neurali ricorrenti sono spesso dotate di "architetture" progettate al preciso scopo di permettere loro la modellazione delle dipendenze di lungo periodo all'interno di un determinato numero di dati, tra le quali, la forma più comunemente utilizzata nelle applicazioni pratiche odierne è, indubbiamente, la c.d. "Long-short term memory" (LSTM) [19, p. 408] [32].

Una LSTM è una tipologia di architettura neurale "feed-back" dove le informazioni sono "immagazzinate" in piccole strutture, definite "cell states", permettendole di gestire

efficacemente volumi di dati sequenziali caratterizzati da una forte "distanza temporale" [19, p. 410] [32]; ciascuna LSTM è inoltre dotata, come si vedrà in seguito, di unità costituite allo scopo di "selezionare" le informazioni maggiormente rilevanti, in entrata e in uscita, per ciascuna "cell state" [19, p. 410] [32].
Più dettagliatamente, ogni "cell state" presente all'interno di una LSTM può essere rappresentata come [19, p. 411] [32]:

$$c_t = c_{t-1} \odot f_t + i_t \odot j_t \tag{4.31}$$

Dove con il simbolo $\odot$ si intende un prodotto matriciale effettuato "elemento per elemento", e, inoltre, rispettivamente:

- c_{t-1} è la "cell state" immediatamente antecedente quella cui si riferisce l'espressione (4.31) [19, p. 410] [32];
- f_t è il c.d. "forget gate" della LSTM, ossia un'unità deputata a decidere quali informazioni contenute in c_{t-1} "immagazzinare" in c_t e quali invece eliminare, operando una trasformazione sigmoidale sui dati di input [19, p. 411] [32]. Questo termine assume, naturalmente, un valore compreso tra 0 e 1, determinando il grado in cui le informazioni contenute in c_{t-1} possano essere utili alla formazione della "cell state" attuale [19, p. 411] [32];
- i_t è il c.d. "input gate" della LSTM, che rappresenta le informazioni "esterne" aggiunte alla "cell state" attuale, c_t, rispetto a quelle "propagate" dalla cella c_{t-1} [19, p. 411] [32];
- j_t è una trasformazione sigmoidale dei dati contenuti nell'input gate, con lo stesso scopo di "filtraggio" dei dati rilevanti per c_t rivestito dal forget gate f_t per il corredo informativo di c_{t-1} [19, p. 411] [32].

Infine, l'output di una generica Long-short term memory può essere rappresentato come [19, p. 411] [32]:

$$h_t = \tanh(c_t) \odot o_t \tag{4.32}$$

Dove o_t è il c.d. "output gate" della LSTM, ossia un'ulteriore trasformazione sigmoidale, del tutto analoga a quelle osservate in precedenza, con lo scopo di "catturare" le informazioni più rilevanti dalla "cell state" attuale c_t.

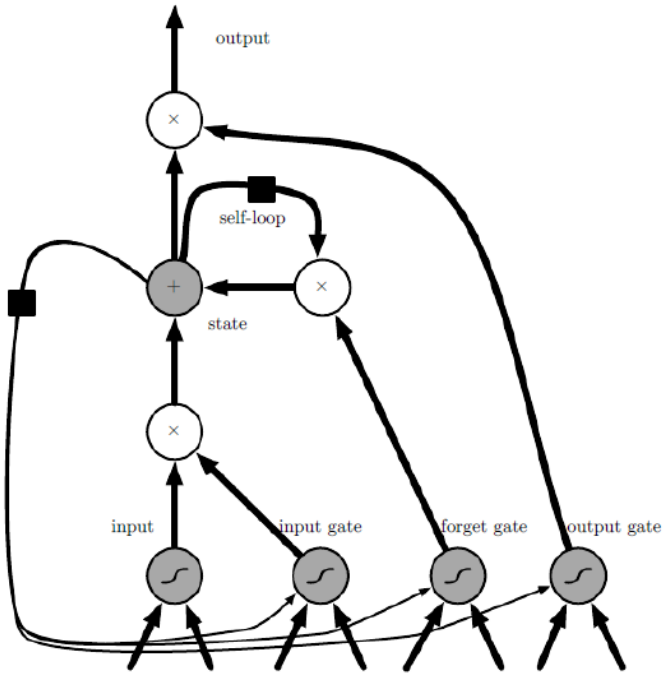

Figura 4.3. Struttura di una Long-short term memory (LSTM) [19, p.409]

Capitolo 5

Collaborative filtering: un confronto empirico tra algoritmi "standard" e reti neurali

I precedenti tre capitoli hanno illustrato i fondamenti teorici del mio lavoro di ricerca personale, discutendo nel dettaglio argomenti di notevole rilevanza, quali i sistemi di raccomandazione e le reti neurali artificiali; il presente capitolo, che mira a concludere la trattazione della tesi in esame, mostrerà come i concetti fin qui descritti possano essere utilizzati in combinazione per costruire avanzati modelli di collaborative filtering, che sappiano raggiungere notevoli livelli di performance, e superare nei risultati gli algoritmi tradizionalmente più utilizzati per la risoluzione di questa tipologia di problemi, quali il k-Nearest Neighbors (k-NN) [11] e la Decomposizione ai valori singolari (SVD) [11].

Più dettagliatamente, la sezione 5.1 descriverà il dataset utilizzato per addestrare gli

algoritmi sviluppati per la ricerca, ossia il noto database "Movielens", nella sua variante "100K" [14]; i due paragrafi successivi, invece, rappresenteranno gli algoritmi maggiormente utilizzati per implementare sistemi di collaborative filtering, ossia k-NN e SVD [11], discutendone i risultati ottenuti in termini di efficacia predittiva sui dati di "testing".

In seguito, le sezioni conclusive esporranno l'approccio "neurale" al collaborative filtering, mostrando i risultati dell'implementazione di una rete neurale "feed-forward" (sia "shallow" che "profonda") ai fini della creazione di un sistema di raccomandazione più accurato dei precedenti; in aggiunta, sarà discussa la possibilità di modellare i dati in questione attraverso un algoritmo sequenziale, come la Long-short term memory, utilizzando una metrica valutativa costruita "ad hoc", nota come "mean reciprocal rank" (MRR) [10].

5.1 Il dataset di riferimento: Movielens 100K

Gli algoritmi destinati alla costruzione di efficaci sistemi di raccomandazione richiedono, usualmente, volumi elevati di dati di "training" per poter essere addestrati con sufficiente accuratezza; a tal scopo, è tendenzialmente preferibile utilizzare dei dataset specializzati, le cui osservazioni siano state raccolte attraverso analisi condotte "ad hoc", piuttosto che sviluppare onerose ricerche su campioni più piccoli e, di conseguenza, meno significativi,

Il dataset più utilizzato nelle applicazioni empiriche, ai fini di generare tecniche di collaborative filtering, è il c.d. "Movielens", il quale è disponibile al download in diversi formati ("100K", "1M", "10M" e "20M"[1]), e raccoglie un ingente volume di recensioni (espresse in forma di ratings, su scala che va da 1 a 5) da parte degli utenti, su un numero elevato di film, segmentati sulla base del genere e della tipologia [14].

[1]Come si può facilmente intuire, le presenti diciture simboleggiano il numero di recensioni presenti all'interno del dataset, che sono, rispettivamente, 100000, 1 milione, 10 milioni e 20 milioni.

Ciascuno dei formati in cui è possibile ottenere il dataset Movielens, è il risultato di ricerche sperimentali condotte, nell'arco di periodi diversi, da GroupLens Research [14]; nello specifico, la prima versione dello stesso, "Movielens 100K", è stata rilasciata nell'aprile del 1998, laddove invece l'ultima, "Movielens 20M", risale all'aprile del 2015, per un totale di 17 anni di "storia", che ne fanno uno dei database più longevi in circolazione, tra quelli deputati alla costruzione di sistemi di raccomandazione [14].
Inoltre, attraverso le osservazioni raccolte ed immagazzinate nei sopracitati datasets, GroupLens ha costituito negli anni il proprio sistema di collaborative filtering, anch'esso chiamato "Movielens" [14]; nell'immagine seguente, è possibile osservare l'evoluzione di tale software dall'anno della sua creazione ad oggi, che si è riflessa in un cambiamento rilevante, tanto nella sua "veste grafica", quanto nella struttura che vi è alla base [14]:

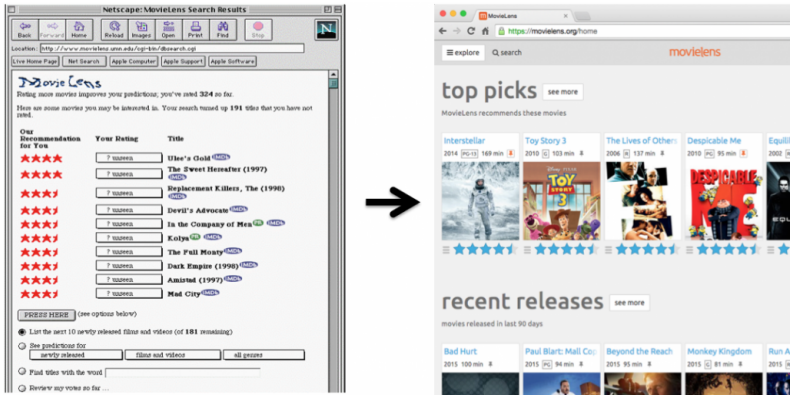

Figura 5.1. Un confronto tra l'aspetto di Movielens nel 1997 e quello odierno [14].

La società di ricerca GroupLens ha inoltre prodotto, periodo dopo periodo, una serie di interessanti statistiche volte a descrivere il comportamento della propria utenza lungo il corso dei propri 17 anni di storia [14]; una di queste, rappresentata dalla figura in basso, è la distribuzione media dei ratings espressi nel suddetto arco cronologico, che, sorprendentemente, tende ad assumere una notevole consistenza rispetto allo scorrere del tempo [14]:

Figura 5.2. Distribuzione dei ratings di Movielens tra il 1997 e il 2015 [14].

Per condurre la mia ricerca personale, ho utilizzato la variante "100K" del dataset Movielens, la quale contiene, come già anticipato in precedenza, 100000 ratings, in scala che va da 1 a 5, espressi da 943 utenti su ben 1682 film [14]; inoltre, i dati in questione sono stati raccolti durante un periodo di sette mesi, dal 19 Settembre del 1997 al 22 Aprile del 1998 [14].
All'interno del dataset in questione sono presenti quattro variabili d'interesse [14]:

- User ID, ossia un numero che identifica lo specifico utente che ha rilasciato il rating [14];
- Movie ID, che, simmetricamente alla variabile precedente, descrive lo specifico film oggetto della recensione [14];
- Rating, ossia il valore numerico, in scala da 1 a 5, del rating espresso da uno specifico utente su un dato film [14];
- Timestamp, ossia un termine numerico che descrive la collocazione "temporale" della recensione [14].

La tabella seguente riassume, a corollario di quanto osservato finora, le prime cinque osservazioni del dataset "Movielens 100K", utilizzato per sviluppare il mio lavoro di ricerca empirica [14]:

User ID	Movie ID	Rating	Timestamp
196	242	3	881250949
186	302	3	891717742
22	377	1	878887116
244	51	2	880606923
166	346	1	886397596

Tabella 5.1. Le prime cinque osservazioni del dataset Movielens "100K" [14]

5.2 L'approccio classico al collaborative filtering: k-Nearest Neighbors (k-NN) e Decomposizione ai valori singolari (SVD)

Nell'introduzione al presente capitolo, si è osservato come le tecniche note con il nome di "k-Nearest Neighbors" (k-NN) e "Decomposizione ai valori singolari" (SVD) rappresentino, ad oggi, un vero e proprio punto di riferimento per la costruzione di sistemi di raccomandazione di tipo "collaborativo" [11] [15, p. 115] [6]; questo paragrafo illustrerà nel dettaglio le generalità di ciascuno di questi algoritmi, per poi descrivere, successivamente, i risultati del mio personale lavoro di ricerca, condotto al fine di rilevare l'efficacia predittiva delle sopracitate tecniche sul "testing set" deputato a tale scopo. Nello specifico, le sezioni 5.2.1 e 5.2.2 saranno incentrate sul metodo k-NN, e completeranno la trattazione riguardante le tecniche di collaborative filtering "Neighborhood-based", illustrate nel secondo capitolo di questa tesi; le sezioni successive, di contro, analizzeranno i metodi di fattorizzazione (o decomposizione) della matrice, focalizzandosi, come già anticipato, sulla SVD, per poi mostrarne un'applicazione empirica alla costruzione di un sistema di raccomandazione reale.

5.2.1 Cos'è il k-NN?

Il k-Nearest Neighbors, noto anche come "k-NN", è uno dei più noti algoritmi di classificazione e regressione utilizzati nel campo dell'apprendimento supervisionato [4, pp. 66-67], e trae il proprio nome, come si è avuto modo di osservare nel paragrafo 2.2.1 di

questa tesi, dal fatto di utilizzare, per la categorizzazione o la predizione dei valori di "testing", le prime k osservazioni più "simili" presenti all'interno del dataset utilizzato per l'addestramento del modello [4, pp. 66-67].

Ma in che modo è possibile determinare la "similarità" tra due o più osservazioni presenti in un "training set", in modo tale da permettere l'implementazione di questo algoritmo?

La misurà di similarità più comunemente utilizzata, in virtù della propria estrema semplicità concettuale, nelle applicazioni del k-NN legate alla creazione di sistemi di raccomandazione, è il c.d. "mean-squared difference" (MSD), che può essere definito come [2, p. 320]:

$$MSD(x,y) = \frac{1}{|I(x,y)|} \cdot \sum_{i \in I(x,y)} [r(x,i) - r(y,i)]^2 \tag{5.1}$$

Dove, nel caso specifico dell'utilizzo di questo algoritmo per costruire un sistema di collaborative filtering, $I(x,y)$ rappresenta un sottoinsieme degli "items" che sono oggetto della raccomandazione, x e y gli utenti di cui è computata la similarità nelle preferenze e, rispettivamente, $r(x,i)$ e $r(y,i)$ i rating attribuiti dall'utente x e dall'utente y sull'oggetto $i \in I(x,y)$ [2, p. 320].

Il MSD assume solo valori ≥ 0, e tende a diminuire nel caso in cui, al contempo, incrementi la similarità tra le osservazioni di x e quelle di y [2, p. 320]; inoltre, nel caso di perfetta adiacenza tra le due variabili in questione, per cui $x = y$, si avrà che MSD$= 0$ [2, p. 320].

Due ulteriori misure di similarità, alternative rispetto al MSD, e che godono di un utilizzo notevole nelle ricerche empiriche, sono la c.d. "similarità del coseno" e il coefficiente di correlazione di Pearson [2, p. 320].

Più specificamente, la prima può essere definita come [2, p. 320]:

$$cos(x,y) = \frac{\sum_{i \in I(x,y)} r(x,i) \cdot r(y,i)}{\sqrt{\sum_{i \in I(x,y)} r(x,i)^2} \cdot \sqrt{\sum_{i \in I(x,y)} r(y,i)^2}} \tag{5.2}$$

Laddove, invece, il coefficiente di Pearson è rappresentato dall'espressione seguente [2, p. 320]:

$$p(x,y) = \frac{\sum_{i \in I(x,y)}[r(x,i) - \bar{x}] \cdot [r(y,i) - \bar{y}]}{\sqrt{\sum_{i \in I(x,y)}[r(x,i)^2 - \bar{x}]} \cdot \sqrt{\sum_{i \in I(x,y)}[r(y,i) - \bar{y}]}} \tag{5.3}$$

Queste due misure presentano notevoli differenze rispetto al MSD, descritto in precedenza: ad esempio, se per il mean-squared difference si può dimostrare, una volta adoperate delle manipolazioni algebriche, che $MSD(x,x) = 0$, rispettando una regola fondamentale del calcolo delle distanze tra variabili[2], le due misure $cos(x,x)$ e $p(x,x)$ sono, contrariamente al principio espresso in precedenza, ambedue pari a 1 [2, p. 321]. Ciò è dovuto, essenzialmente, al fatto che entrambe le metriche non abbiano un range di valori che va da 0 a 1, ma da -1 a 1, per cui, in maniera del tutto opposta rispetto a quanto osservato per il MSD, la similarità tra due elementi x e y sarà maggiore all'aumentare dei valori di $cos(x,y)$ e $p(x,y)$ [2, p. 321].

Una volta determinata la misura attraverso cui computare la similarità tra le osservazioni presenti nel dataset di "training", il passo successivo per l'implementazione dell'algoritmo k-NN è, come facilmente intuibile, la scelta del parametro k, rappresentante il numero di "neighbors", ossia delle osservazioni più "vicine", in relazione all'indice di similarità utilizzato, di cui tener conto ai fini della classificazione o della predizione dei valori presenti nel dataset di "testing" [4, p. 70].

Il valore di k dovrà essere selezionato tenendo conto di quello che, nella teoria dell'apprendimento automatico, è definito "bias-variance tradeoff", ossia l'equilibrio esistente tra "overfitting" e "underfitting"[3] del modello, rispetto ai dati utilizzati per addestrarlo [4, p. 70]. Valori eccessivamente alti di k, infatti, possono ridurre l'impatto della dispersività dei dati in esame sulle performance mostrate dal modello, ma al contempo

[2]Secondo cui, come anticipato, la distanza tra un termine e sè stesso deve essere necessariamente nulla.

[3]Per "overfitting" si intende, nelle applicazioni connesse al Machine Learning, il problema legato a modelli che si adattano eccessivamente ai dati di "training", e possiedono per tale ragione scarse capacità di generalizzazione dei propri risultati; di contro, il problema opposto, detto anche "underfitting", designa l'incapacità, da parte di un modello, di cogliere le complessità presenti nei dati di "training", finendo in tal modo per essere eccessivamente semplice e "generico".

non permettono all'algoritmo di analizzare con sufficiente cautela taluni "pattern" di minore entità, ma di indubbia rilevanza, presenti nel dataset [4, pp. 70-71]; di contro, la scelta di un termine k basso, può portare ad un'influenza decisamente più ampia della variabilità dei dati sui risultati ottenuti dal k-NN [4, p. 71].
Come si può intuire, la scelta ottima del parametro k da utilizzare per l'implementazione del k-Nearest Neighbors, dovrà necessariamente ricadere su un valore mediano rispetto agli estremi descritti in precedenza [4, p. 71]; come si osserverà nella prossima sezione, dove sarà esposto il mio lavoro di ricerca in materia, è possibile effettuare diverse prove "sperimentali" a tal scopo, osservando le variazioni della perdita connessa al modello in relazione alle fluttuazioni di k.

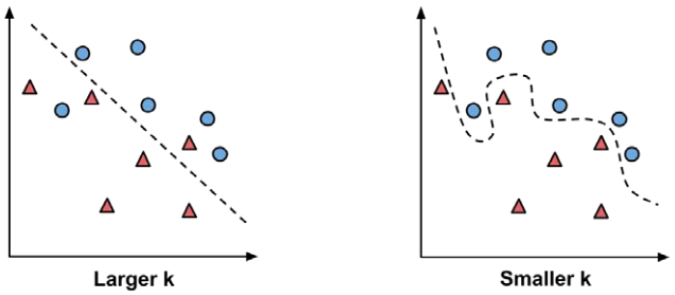

Figura 5.3. Descrizione del "bias-variance tradeoff" nella scelta del parametro k [4, p. 71].

5.2.2 k-NN e collaborative filtering: descrizione della ricerca e analisi dei risultati

In questa sezione, discuterò gli esiti del mio lavoro di ricerca sperimentale riguardante l'utilizzo dell'algoritmo k-NN per l'implementazione di un sistema di collaborative filtering; questo studio, così come quelli che seguiranno nei paragrafi successivi, è stato condotto sul database "Movielens 100K", come già anticipato all'inizio del presente capitolo.
Nello specifico, sono state indagate le performance dell'algoritmo in relazione a tre valori del parametro k: 32, 50 e 100; inoltre, sono state utilizzate, come misure di

similarità per l'identificazione dei "nearest neighbors", tutte e tre le metriche osservate nel precedente paragrafo (MSD, similarità del coseno e indice di Pearson).
L'indice utilizzato per quantificare la "perdita" connessa al modello in esame è il c.d. "root-mean squared error" (RMSE)[4], una variante del già analizzato mean-squared error, la cui rappresentazione matematica è la seguente:

$$\text{RMSE} = \sqrt{\text{MSE}} = \sqrt{\frac{1}{2n} \cdot \sum_{x} [\hat{y}(x) - y]^2} \tag{5.4}$$

Dove, nel caso specifico in esame, y rappresenta, per ciascun input x, il valore predetto dal k-Nearest Neighbors, mentre $\hat{y}(x)$ non è altro che il rating realmente espresso da ciascun utente all'interno del dataset di "testing".
Sulla base delle informazioni appena fornite, è possibile riassumere i risultati[5] della ricerca in questione, attraverso la seguente tabella:

k	Similarità	RMSE
32	Coseno	1.0226
50	Coseno	1.0219
100	Coseno	1.0225
32	Pearson	1.0205
50	Pearson	1.0156
100	Pearson	1.0158
32	MSD	0.9813
50	MSD	0.9862
100	MSD	0.9973

Tabella 5.2. Riassunto dei risultati della ricerca condotta attraverso il k-NN.

Come si può agevolmente osservare, le performance migliori risultano essere ottenute attraverso l'utilizzo del MSD come misura di similarità; di contro, la metrica relativa al coseno e l'indice di Pearson presentano risultati decisamente peggiori, a prescindere

[4]Come si avrà modo di osservare nei successivi paragrafi, la stessa metrica è stata utilizzata per valutare le performance di ciascun algoritmo analizzato in questo capitolo, al fine di ottenere una comparazione più agevole tra i rispettivi risultati.

[5]Per una descrizione maggiormente dettagliata delle modalità di conduzione dell'indagine, si rimanda al primo paragrafo dell'appendice posta a conclusione di quest'elaborato.

dal valore attribuito al parametro k.

In termini assoluti, la perdita minore connessa all'utilizzo del k-NN (0.9813) è ottenuta per un valore di $k = 32$ e utilizzando come metrica di similarità il MSD; inoltre, come si può osservare analizzando la tabella, utilizzando questa specifica misura di similarità, sembrerebbe esserci una forte correlazione positiva tra il valore k e il RMSE, in quanto, a variazioni in aumento del primo termine, sono connessi, in maniera del tutto speculare, incrementi nel secondo valore.

Ciò mi ha portato a concludere che, per il tipo di ricerca in questione, fosse preferibile limitare la scelta di k a valori sufficientemente bassi, in virtù del fatto che ad essi fosse connessa una perdita complessiva del modello decisamente più "contenuta".

Il grafico successivo riassume quanto detto finora, descrivendo l'andamento delle performance (quantificate dal RMSE) relative all'algoritmo k-NN, per diversi livelli del parametro k e in rapporto alle tre misure di similarità osservate in precedenza:

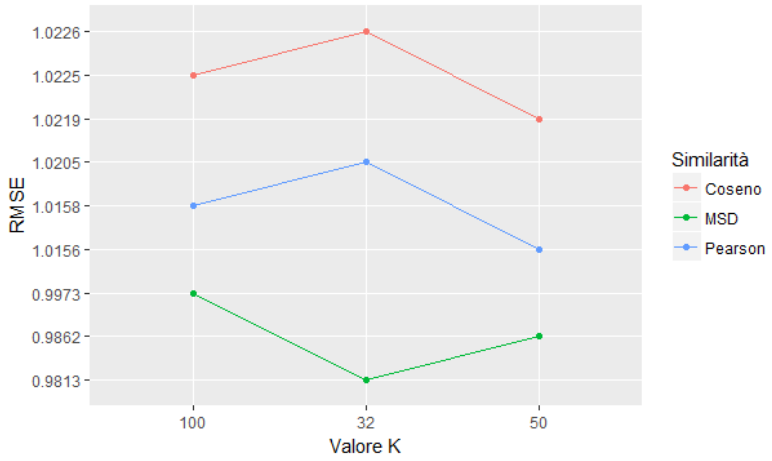

Figura 5.4. Rappresentazione grafica dei risultati della ricerca condotta attraverso il k-NN.

5.2.3 Fattorizzazione della matrice e SVD

Le tecniche di decomposizione (o fattorizzazione) della matrice costituiscono, come già osservato in apertura di capitolo, una delle principali famiglie di algoritmi utilizzati per la costruzione di sistemi di collaborative filtering [41, p. 43]; nello specifico, questi

strumenti mirano a ridurre la complessità di una matrice iniziale, che descriva l'architettura originaria dei dati, "decomponendola" nel prodotto di più matrici, ciascuna rappresentante un dato numero di "fattori latenti", che ne esplicitino le strutture nascoste alla base [41, p. 43] [6, p. 140].
Tra le tecniche di fattorizzazione della matrice, la più utilizzata nelle applicazioni pratiche legate ai sistemi di raccomandazione è, indubbiamente, quella nota con il nome di "Decomposizione ai valori singolari", o, più brevemente, SVD (dall'originale accezione di "Singular value decomposition") [41, p. 44] [6, p. 140]; più dettagliatamente, l'algoritmo SVD prevede che, data una matrice A dalle dimensioni $m \times n$, questa possa essere "scomposta" nel seguente prodotto [6, p. 140]:

$$A = U \cdot \Sigma \cdot V^T \tag{5.5}$$

Dove, rispettivamente [6, p. 140]:

- U è una matrice ortogonale[6], di dimensioni $m \times m$ [6, p. 141];
- Σ è una matrice diagonale[7], i cui elementi posti sulla diagonale principale, $\sigma_1, \sigma_2, \sigma_3$, ..., σ_n, sono detti i "valori singolari" di A [6, p. 141];
- V^T è la trasposta[8] di una matrice ortogonale V, dalle dimensioni $n \times n$ [6, p. 141].

Come facilmente intuibile, l'algoritmo di SVD è utilizzato, nelle applicazioni reali di collaborative filtering, allo scopo di "ricostruire" la matrice user-rating, che, come si è visto nel secondo capitolo di questo elaborato, rappresenta un fondamentale strumento di analisi collegato all'implementazione di sistemi di raccomandazione di questa tipologia [41, p. 44].

[6] In algebra, una matrice M, di dimensioni $n \times n$, si dice "ortogonale" se il prodotto tra la matrice stessa e la sua trasposta, $M \cdot M^T$, è uguale ad una matrice identità delle medesime dimensioni, I [6, p. 141].

[7] Nell'algebra matriciale, una matrice si dice "diagonale" se tutti i suoi elementi, ad eccezione di quelli posti sulla diagonale principale, sono pari a 0.

[8] In algebra, una matrice M^T si dice "trasposta" di una matrice M, se è ottenuta scambiandone le righe con le colonne.

Tuttavia, la sparsità di questa matrice, composta per lo più da valori mancanti, rappresenta, nella pratica, un ostacolo di notevole rilevanza per l'uso di tecniche di decomposizione matriciale come quella descritta in precedenza [41, p. 44]; a tal proposito, diversi modelli alternativi rispetto alla versione "tradizionale" della SVD, capaci di porre un rimedio alla criticità sopracitata, sono stati proposti in letteratura, tra i quali vale indubbiamente la pena citare il contributo offerto nel 2006 da un partecipante al "premio Netflix", noto con lo pseudonimo di "Simon Funk", e già citato nella sezione 2.2.1 di questa tesi [8] [41, p. 44].
Il modello di SVD proposto da "Simon Funk", nello specifico, prevede che ciascuna predizione $\hat{r}_{ui}$ del rating fornito da un generico utente u su un dato oggetto i, possa essere rappresentata come il prodotto scalare di due matrici, come segue [41, p. 44] [8] [11]:

$$\hat{r}_{ui} = q_i^T \cdot p_u \tag{5.6}$$

Dove q_i^T e p_u descrivono due "embeddings", che esprimono i "fattori latenti" rappresentanti, rispettivamente, le caratteristiche dell'oggetto i e quelle dell'utente u, che hanno condotto alla formazione di un determinato rating atteso $\hat{r}_{ui}$ [41, p. 44] [8] [11].
Tale algoritmo prevede che i due sopracitati "embeddings" possano essere "ricostruiti", partendo da una scelta iniziale casuale dei valori di q_i^T e p_u, e utilizzando successivamente l'algoritmo di discesa del gradiente, o in alternativa la SGD (già descritte in precedenza all'interno del quarto capitolo), per minimizzare una determinata funzione di costo [41, p. 44] [8] [11], che, nel caso specifico del modello di SVD ivi proposto, è data da [41, p. 44] [8] [11]:

$$\sum_{(u,i)\in K} (r_{ui} - \hat{r}_{ui})^2 + \lambda \cdot (||q_i||^2 + ||p_u||^2) \tag{5.7}$$

Tale funzione è nota anche con il nome di "errore quadratico regolarizzato", o, seguendo la notazione originale, "regularized-squared error", e rappresenta una "variante" delle comuni funzioni quadratiche già osservate in precedenza, rispetto alle quali aggiunge un termine di "regolarizzazione" λ, con lo scopo di migliorare l'efficienza generale del

modello, evitando, per quanto possibile, che si generi un "overfitting" sui dati di "training" [41, p. 44].
Il modello "alternativo" di decoposizione ai valori singolari, ivi proposto, rappresenta ad oggi uno degli strumenti maggiormente utilizzati, nelle ricerche empiriche, allo scopo di implementare delle avanzate architetture di collaborative filtering [41, p. 44]; anche il mio studio personale in materia, che sarà esposto nel paragrafo seguente, è basato su questa versione dell'algoritmo di SVD, decisamente più "complessa" e articolata rispetto alla sua controparte "tradizionale", ma al contempo maggiormente accurata nella formulazione delle proprie predizioni [41, p. 44].

5.2.4 SVD e collaborative filtering: decrizione della ricerca e analisi dei risultati

Il presente paragrafo discuterà i risultati della mia ricerca personale, condotta allo scopo di valutare l'efficacia dell'algoritmo SVD (nella "variante" proposta da "Simon Funk") nella costruzione di un sistema di raccomandazione di tipo "collaborativo".
Nello specifico, ho analizzato la capacità predittiva del modello, simmetricamente a quanto osservato per l'algoritmo k-NN, utilizzando come metrica di valutazione il RMSE; inoltre, ho fissato arbitrariamente il numero di fattori latenti componenti gli "embeddings", q_i^T e p_u, a 32.
Per computare la perdita generata dal modello, come anticipato nel paragrafo precedente, è stata utilizzata la funzione di costo nota come "regularized-squared error", minimizzata attraverso il ricorso all'algoritmo di discesa stocastica del gradiente (SGD), per un totale di 50 "epoche" di addestramento.
La seguente tabella illustra i risultati[9] della ricerca, ottenuti successivamente alla fase di "training" del modello sul dataset "Movielens 100K", per valori del learning rate η pari, rispettivamente, a 0.001, 0.01 e 0.1:

[9]Per una descrizione maggiormente dettagliata delle modalità di conduzione dell'indagine, si rimanda al secondo paragrafo dell'appendice posta a conclusione di quest'elaborato.

η	Fattori latenti	Epoche	Metodo di ottimizzazione	RMSE
0.001	32	50	SGD	0.9809
0.01	32	50	SGD	1.0291
0.1	32	50	SGD	1.3016

Tabella 5.3. Riassunto dei risultati della ricerca condotta attraverso la SVD.

Come si può agevolmente osservare, analizzando la tabella, il RMSE segue un forte andamento crescente rispetto alle variazioni nel learning rate η, mostrando una notevole correlazione positiva tra le due variabili sopracitate: infatti, il risultato migliore in termini di minimizzazione del RMSE, ossia 0.9809, è ottenuto utilizzando un valore di η pari a 0.001, laddove, invece, l'utilizzo di parametri più alti del learning rate ha condotto ad esiti finali decisamente meno favorevoli agli scopi prefissati dal mio lavoro di ricerca.

Il seguente grafico esemplifica ancor più la consistenza dei risultati descritti in precedenza, fornendo una panoramica chiara sull'andamento del RMSE al variare del parametro η del modello di SVD:

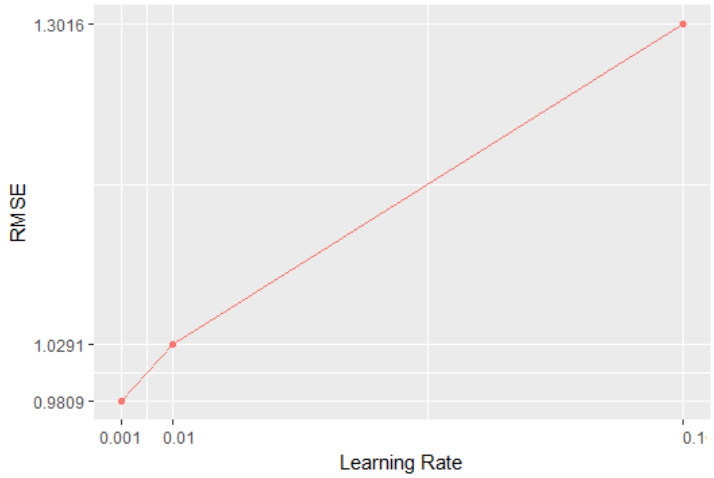

Figura 5.5. Rappresentazione grafica dei risultati della ricerca condotta attraverso la SVD.

5.3 L'approccio "neurale" al collaborative filtering: Reti "feed-forward" e LSTM

Le sezioni precedenti hanno mostrato i risultati parziali della mia ricerca empirica, condotta ai fini di analizzare il rendimento (attraverso l'analisi di un indice sintetico, noto come RMSE) di due tecniche "tradizionalmente" utilizzate nell'ambito del collaborative filtering, ossia il k-NN e la SVD.
Il presente paragrafo mira a completare la panoramica sul mio studio personale in materia, analizzando i risultati di un lavoro analogo, effettuato mediante il ricorso alle reti neurali artificiali, descritte nel terzo e quarto capitolo di questo elaborato: nel dettaglio, la sezione 5.3.1 analizzerà le performance raggiunte mediante l'utilizzo di due reti neurali "feed-forward" (una, appartenente alla tipologia "shallow", e l'altra, maggiormente "profonda"), ai fini dell'implementazione di un sistema di raccomandazione "collaborativo"; di contro, invece, la sezione 5.3.2 discuterà la possibile costruzione di un modello "sequenziale", allo scopo di descivere appropriatamente il "comportamento" dei dati utilizzati per la fase di "training", attraverso il ricorso ad un'architettura neurale di tipo Long-short term memory (LSTM).

5.3.1 Reti "feed-forward" e collaborative filtering: descrizione della ricerca e analisi dei risultati

I capitoli 3 e 4 di questa tesi hanno descritto i fondamenti teorici delle reti neurali artificiali; nella presente sezione, tale "theoretical framework" sarà utilizzato per discutere i risultati della mia ricerca personale, volta alla costruzione, in maniera del tutto speculare a quanto visto in precedenza per il k-NN e la SVD, di un ulteriore modello (in questo caso, una rete neurale "feed-forward") capace di fornire un'architettura di base per la costruzione di un sistema di collaborative filtering.
In dottrina è comunemente utilizzato un approccio "misto" alla costruzione di modelli "neurali" che permettano di implementare sistemi di raccomandazione, il quale, nello

specifico, fa uso del concetto di "embedding", già discusso in precedenza nella trattazione relativa all'algoritmo di SVD [40]; nel dettaglio, il punto di partenza per il mio studio individuale, è stata la creazione di due "embeddings" che rappresentassero, rispettivamente, le caratteristiche "latenti" relative agli utenti e agli oggetti delle recensioni presenti nel dataset "Movielens 100K", utilizzando un numero di fattori pari a 32.

I livelli della rete risultanti da questa operazione, definibili strati di "embedding" (embedding layers) [40], non costituiscono altro che la prima trasformazione, presente all'interno del modello, dei dati originari di input.

L'output degli strati di "embedding", nel caso specifico in esame, è un vettore bidimensionale di grandezza (1, 32); ai fini della propagazione del risultato negli strati successivi della rete, si è ritenuto necessario operare una trasformazione ulteriore di questi valori, riducendo la dimensionalità degli stessi da 2 a 1.

Ciò è stato possibile inserendo due strati di "flatten" (flatten layers), immediatamente successivi agli strati di "embedding", con l'unico scopo di ridurre dimensionalmente l'output generato da questi ultimi, da un vettore di due valori (1, 32), ad un unico insieme "compresso" di 32 neuroni[10].

Un ulteriore trasformazione dei dati in questione è stata applicata al livello successivo, dove ciascuno dei due strati neurali di "flatten" è stato sottoposto ad un'operazione, molto comune nelle ricerche applicate in materia, nota come "dropout" [27, p.1930]; tale procedura, utilizzata allo scopo di prevenire casi di "overfitting" nelle reti neurali, prevede che, durante ciascun ciclo, o "epoca", di addestramento, sia rimossa una determinata quota (nel mio caso, pari a 0.2) dei neuroni contenuti in uno specifico strato del modello, che nell'applicazione in esame è stato definito strato di "dropout" (dropout layer) [27, p.1930].

Utilizzando tale procedura, è stato possibile costruire due strati di "dropout", i quali,

[10]Nello specifico, quest'operazione di "appiattimento" dell'output è stata effettuata moltiplicando tra loro gli elementi di ambedue i vettori degli strati di "embedding", ottenendo per entrambi una dimensione finale pari a $1 \cdot 32 = 32$.

in una fase successiva, sono stati concatenati in un unico strato di "unione" (merged layer), passando da due livelli separati, ciascuno contenente 32 neuroni, ad un unico, singolo "layer", da 64 neuroni complessivi. L'output del "merged layer" è stato infine "propagato" attraverso due strati "fully connected", il primo dei quali avente come proprio output un numero di neuroni pari a 200, ed il secondo comprendente un unico neurone, rappresentante la predizione finale del rating generata dal modello.
Per ciascuno dei due strati finali "fully connected", la funzione di attivazione utilizzata è stata di tipo "Rectified Linear" (ReLu), in modo tale da convertire ogni possibile termine negativo presente in tali "layers", inutile ai fini della ricerca in esame, in un neurone avente valore maggiore o uguale a 0.
Lo schema seguente esemplifica la struttura del modello di rete neurale fin qui descritto:

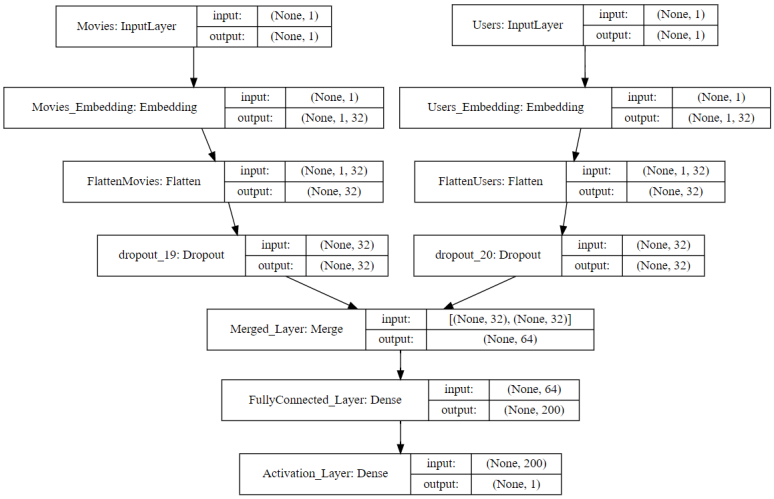

Figura 5.6. Schema del modello di rete neurale "shallow" discusso nelle righe precedenti.

Il modello di rete neurale precedentemente descritto, è, nonostante possa apparire a prima vista estremamente articolato e complesso, afferente alla tipologia di reti "shallow" discusse nel terzo capitolo di questo elaborato; infatti, le operazioni condotte in ciascuno strato antecedente al primo "fully connected layer", sono state unicamente finalizzate ad un "pre-processing" dei dati di input, allo scopo di renderli utilizzabili

dal modello per gli scopi definiti dalla ricerca in esame.

Questo modello di rete neurale "shallow" è stato addestrato attraverso l'algoritmo di discesa stocastica del gradiente (SGD), utilizzato per minimizzare la funzione di costo prescelta, che, nel caso specifico in esame, è stata l'errore quadratico medio (MSE), già ampiamente discusso nelle precedenti sezioni di questa tesi.

La tabella successiva riporta gli esiti[11] della ricerca in questione, per valori del learning rate η pari, rispettivamente, a 0.001, 0.01 e 0.1:

η	Fattori latenti	Epoche	Dropout	Metodo di ottimizzazione	RMSE
0.001	32	50	0.2	SGD	0.9816
0.01	32	50	0.2	SGD	0.9624
0.1	32	50	0.2	SGD	0.9911

Tabella 5.4. Riassunto dei risultati della ricerca condotta attraverso il modello di rete neurale "shallow".

Come si può agevolmente osservare, analizzando la tabella, le performance generali del modello di rete neurale risultano tendenzialmente migliori rispetto a quelle mostrate dagli algoritmi k-NN e SVD, con un "picco" complessivo in corrispondenza del valore di $\eta = 0.01$, per cui il RMSE del modello vale 0.9624, il minor valore dell'indice fin qui registrato.

Tuttavia, proprio una performance a tal punto positiva del modello in esame, mi ha condotto ad indagare i possibili esiti dell'implementazione di una rete maggiormente "profonda", con ben tre strati "nascosti", tutti rigorosamente "fully conncected", antecedenti il livello finale di output, per la risoluzione del problema di collaborative filtering in oggetto.

Nello specifico, il primo degli strati in questione, ha applicato una trasformazione dei dati in ingresso del tutto analoga all'unico "fully connected layer" del modello precedente; i due livelli successivi, invece, sono stati utilizzati allo scopo di "ridurre" progressivamente il numero di neuroni "propagati" da ciascuno strato intermedio a quello

[11] Per una descrizione maggiormente dettagliata delle modalità di conduzione dell'indagine, si rimanda al terzo paragrafo dell'appendice posta a conclusione di quest'elaborato.

finale della rete, come il seguente schema illustra chiaramente:

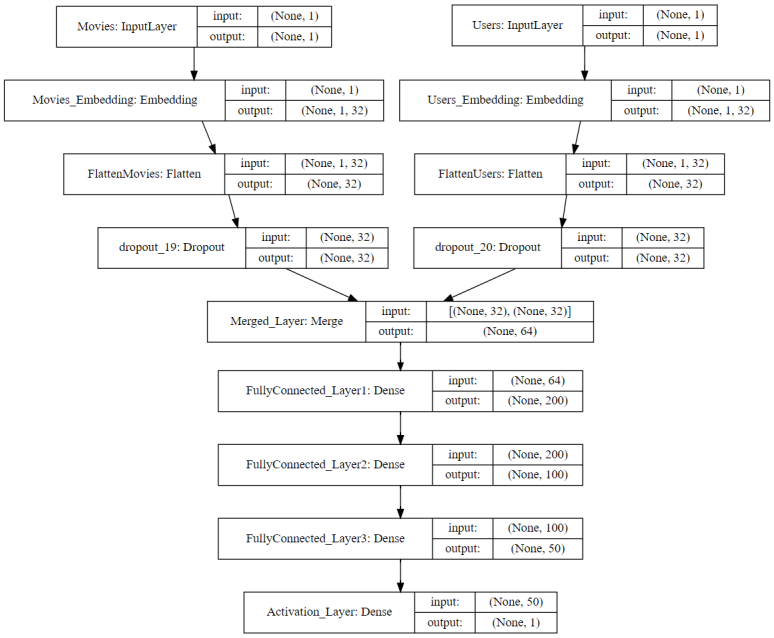

Figura 5.7. Schema del modello di rete neurale "profonda" discusso nelle righe precedenti.

Per ciascuno dei "fully connected layers" rappresentati nello schema precedente, la funzione di attivazione utilizzata è stata della tipologia "ReLu", per le stesse ragioni esplicate relativamente al suo utilizzo per il modello di rete neurale "shallow".
Quest'ultimo modello di rete neurale è stato addestrato con le medesime modalità del primo, attraverso l'algoritmo di SGD applicato per la "discesa" della funzione di costo rappresentata dal MSE; la seguente tabella illustra, dettagliatamente, i risultati raggiunti in termini di capacità predittiva attraverso il ricorso a questo complesso strumento di analisi:

η	Fattori latenti	Epoche	Dropout	Metodo di ottimizzazione	RMSE
0.001	32	50	0.2	SGD	0,98
0.01	32	50	0.2	SGD	0.9593
0.1	32	50	0.2	SGD	1.01

Tabella 5.5. Riassunto dei risultati della ricerca condotta attraverso il modello di rete neurale "profonda".

Come si può agevolmente osservare dalla tabella, le performance di quest'ultimo modello risultano essere tendenzialmente superiori anche all'architettura di rete neurale "shallow" analizzata in precedenza (se si eccettua, naturalmente, il valore del RMSE in corrispondenza di $\eta = 0.1$); nello specifico, la rete neurale ivi presentata mostra il risultato complessivamente migliore rispetto a tutte le altre tecniche algoritmiche utilizzate in precedenza, ossia un RMSE pari a 0.9593 in corrispondenza di un valore del learning rate pari a 0.01.
La seguente immagine sintetizza il confronto tra le due architetture di rete neurale utilizzate all'interno di questo lavoro di ricerca, evidenziandone l'andamento del RMSE per diversi valori del learning rate η:

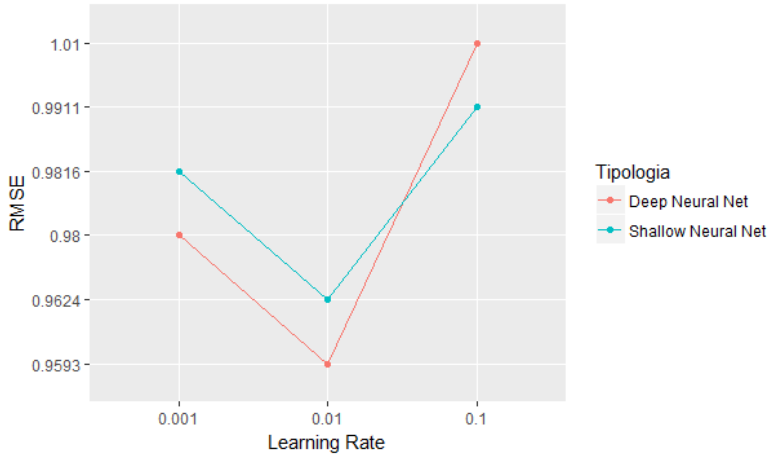

Figura 5.8. Confronto tra le performance dei due modelli di rete neurale descritti in precedenza.

5.3.2 LSTM e collaborative filtering: descrizione della ricerca e analisi dei risultati

L'ultimo "passo" effettuato nell'ambito del mio studio personale è stato l'adozione di un modello di rete neurale "feed-back", della tipologia LSTM (già analizzato nei suoi fondamenti teorici, nella sezione conclusiva del precedente capitolo), allo scopo di valutare il possibile utilizzo di un algoritmo "sequenziale" per la modellazione dei dati contenuti

all'interno di "Movielens 100K"; nello specifico, ai fini della costruzione del modello di rete ricorrente sopra descritto, è stato utilizzato un approccio di "preparazione" dei dati iniziali di input simile a quello osservato per il modello "feed-forward", attraverso la creazione di due strati di "embedding", ciascuno contenente 32 strutture latenti.
Successivamente, il modello è stato addestrato, come le precedenti architetture neurali e la SVD, attraverso l'algoritmo di discesa stocastica del gradiente (SGD), applicato per la minimizzazione di una funzione di costo già consolidata, nella letteratura esistente in materia, per la risoluzione di problemi di tale tipologia, nota come "Bayesian personalized ranking" (BPR) [3, p. 5]; nello specifico, tale funzione è utilizzata per comparare lo "score" registrato da un oggetto i cui è stato attribuito un rating positivo, con un campione n_s di oggetti j aventi rating negativo, utilizzando il valore medio di questi ultimi per approssimare la perdita generata dal modello, come mostra la seguente espressione [3, p. 5]:

$$\text{BPR} = -\frac{1}{n_s} \cdot \sum_{j=1}^{n_s} log(\sigma(\hat{r}_{s,i} - \hat{r}_{s,j})) \tag{5.8}$$

Infine, la performance complessiva del modello sequenziale di tipo LSTM è stata valutata attraverso un indice sintetico costruito "ad hoc", noto anche come "ranking medio reciproco", o MRR (dall'originale "mean reciprocal rank") [3, p. 6] [10]; questa metrica, particolarmente utilizzata in ambito dell'IR (information retrieval), misura la media ponderata dei reciproci dei tentativi effettuati prima di giungere alla soluzione corretta di una determinata "query" [26].
Nel caso specifico del modello di LSTM in esame, ciascuna "sequenza" ordinata di ratings all'interno del "testing set" è stata divisa in due parti, una contenente l'ultimo giudizio espresso, e l'altra, che racchiudeva tutti gli altri [10]; il MRR è stato applicato, al tal proposito, per analizzare la capacità dell'algoritmo in questione di predire l'ultimo rating della sequenza attraverso l'utilizzo dei precedenti, quantificando la sua "efficacia" in termini del valore reciproco del numero di tentativi effettuati prima di giungere alla predizione corretta [10].

A valori più bassi del MRR[12] corrisponde, come facilmente intuibile, una minore efficacia predittiva del modello (e viceversa); nel caso specifico in esame, i risultati[13] mostrati dal modello di LSTM per valori del learning rate pari a 0.001, 0.01 e 0.1, utilizzando un MRR "medio" tra le diverse sequenze di valori nel dataset di "testing", sono riassunti dalla seguente tabella:

η	Fattori latenti	Epoche	Metodo di ottimizzazione	Average MRR
0.001	32	50	SGD	0.0268
0.01	32	50	SGD	0.049
0.1	32	50	SGD	0.0217

Tabella 5.6. Riassunto dei risultati della ricerca condotta attraverso il modello di rete LSTM.

Come si può agevolmente osservare, in ciascuno dei tre casi analizzati, il valore medio del MRR è risultato eccessivamente basso, di poco maggiore rispetto al suo limite inferiore, ossia 0; ciò mi ha portato a concludere che, ai fini dell'implementazione di un sistema di collaborative filtering, non risulti particolarmente efficace l'utilizzo di un modello sequenziale come la rete LSTM descritta in questa sezione, mostrando essa una performance notevolmente bassa, se rapportata agli altri algoritmi (k-NN, SVD e reti neurali "feed-forward") analizzati in precedenza.

5.4 Approcci a confronto e esiti della ricerca

Gli esiti della ricerca esposta in questo capitolo hanno evidenziato, sostanzialmente, come un approccio al collaborative filtering basato sull'utilizzo di reti neurali artificiali, possa presentare delle performance notevoli, persino superiori a quelle mostrate da algoritmi tradizionalmente impiegati a tal scopo, come il k-NN e la SVD.

[12] Il quale assume valori compresi tra 0 e 1, dove 0 indica che il modello non ha fornito alcuna predizione corretta, e 1 che l'algoritmo ha individuato correttamente l'ultimo rating della sequenza al primo tentativo.

[13] Per una descrizione maggiormente dettagliata delle modalità di conduzione dell'indagine, si rimanda al quarto paragrafo dell'appendice posta a conclusione di quest'elaborato.

Di contro, la precedente sezione ha analizzato la sostanziale inefficacia di un modello sequenziale, costruito attraverso una rete LSTM, ai fini dell'implementazione di un sistema di raccomandazione efficiente, in virtù delle performance negative da esso mostrate, in termini di valori del MRR notevolmente inferiori alle attese iniziali.
Comparando, nello specifico, i risultati di maggiore rilevanza ottenuti, nelle analisi precedentemente esposte, dai modelli k-NN, SVD e di rete neurale "feed-forward", sia "shallow" che profonda, si può costruire una rappresentazione sintetica degli esiti generali della mia ricerca personale, come mostrato dalla seguente tabella:

Modello	RMSE
k-NN	0.9813
SVD	0.9809
Rete neurale "shallow"	0,9624
Rete neurale "profonda"	0.9593

Tabella 5.7. Rappresentazione sintetica degli esiti della mia ricerca personale.

Tale tabella mostra due "findings" essenziali derivabili dal mio studio individuale, precedentemente esposto: il primo è, come agevolmente osservabile, la "superiorità" dell'approccio "neurale" alla raccomandazione, testimoniato da valori del RMSE sensibilmente più bassi rispetto a quelli osservati per le metodologie k-NN e SVD; di contro, il secondo risultato che può essere dedotto dalla rappresentazione precedente, è la notevole performance mostrata dal modello di rete neurale "profonda", il quale, con un RMSE pari a 0.9593, rappresenta indubbiamente il metodo maggiormente efficace, tra quelli proposti, in termini di capacità predittiva, ponendo le basi per una possibile introduzione, all'interno di tale complesso campo di studi, di un nuovo ed innovativo approccio alla raccomandazione, interamente basato sul "deep learning".
Il seguente grafico illustra quanto detto in precedenza, evidenziando le variazioni dell'errore predittivo sul dataset "Movielens 100K", rappresentato dal RMSE, per diversi livelli relativi all'algoritmo utilizzato per la modellazione dei dati di "training":

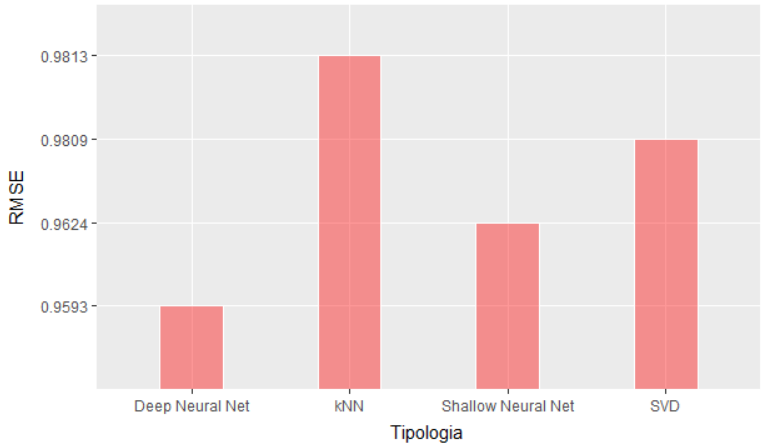

Figura 5.9. Rappresentazione grafica dei risultati finali del mio studio individuale.

Capitolo 6

Conclusioni

Il capitolo precedente ha esposto i risultati fondamentali del lavoro di ricerca, da me condotto, sul corretto utilizzo di strumenti algoritmici adeguati alla costruzione di un sistema di raccomandazione di tipo "collaborativo"; nel dettaglio, come desumibile dal contenuto delle sezioni precedenti, la trattazione si è focalizzata sull'utilizzo delle reti neurali artificiali, e in special modo dei modelli di natura "feed-forward", allo scopo di costruire degli strumenti avanzati di marketing "one-to-one", come, per l'appunto, i sistemi di raccomandazione in esame.

Gli esiti dello studio in questione hanno mostrato la tendenziale "superiorità" di un approccio "neurale" al collaborative filtering, rispetto alle tecniche utilizzate nelle applicazioni "tradizionali" in materia, ossia il k-NN e la SVD; inoltre, un ulteriore "finding" di notevole rilevanza, associato a questo lavoro di ricerca, è il ruolo rivestito dalle reti neurali "profonde", le quali hanno presentato le migliori performance, in termini di capacità predittiva, rispetto a ciascuno degli algoritmi descritti nei paragrafi precedenti.

Tutto ciò mi ha condotto ad osservare che l'utilizzo di modelli "deep-learning-based" potrebbe essere esteso oltre il raggio del collaborative filtering, conducendo ad applicazioni empiriche degli stessi per la costruzione di sistemi di natura "content-based", oltre che per l'implementazione di architetture di raccomandazione "ibride"; il risultato

di un ampliamento dei campi d'uso dei modelli di rete neurale "profondi" potrebbe essere, nel lungo periodo, un complessivo rinnovamento delle pratiche vigenti in materia, oltre che una notevole innovazione per l'intera sfera delle soluzioni legate al marketing personalizzato "2.0".

Tuttavia, bisogna tener presente che la portata dei risultati ivi discussi è in ogni caso limitata agli algoritmi in base ai quali è stato operato il confronto; per tale ragione, ai fini di ottenere una generalizzazione maggiormente significativa degli esiti in questione, potrebbero essere condotti degli studi ulteriori, focalizzati sull'utilizzo di tecniche alternative a quelle descritte nei precedenti capitoli, come modelli di rete neurale di diversa tipologia (ad esempio, le architetture "convoluzionali"), oppure strumenti radicalmente diversi, ma di comprovata efficacia, nella risoluzione di problemi legati alla raccomandazione.

Infine, tentativi ulteriori potrebbero essere effettuati in riferimento all'utilizzo di modelli di natura "sequenziale", diversi dalla LSTM discussa in questa sede, in modo tale da comprendere se le ragioni sottostanti i risultati ottenuti, denotanti una quasi totale assenza di capacità predittiva del modello, risiedano nella scelta specifica dell'algoritmo applicato alla risoluzione del caso in esame, o piuttosto ad una tendenziale inadeguatezza, da parte di architetture di questa tipologia, nel "modellare" efficacemente l'insieme di informazioni necessarie alla costruzione di un sistema di raccomandazione efficiente.

Appendice A

Lista dettagliata dei programmi e degli algoritmi

Questa appendice raccoglie, come da titolo, l'insieme dei programmi scritti in linguaggio Python per l'implementazione degli algoritmi usati nel lavoro di ricerca illustrato nella tesi.

A.1 k-NN

Per implementare l'algoritmo k-Nearest Neighbors ho utilizzato Surprise [11], un utile scikit[1] costruito su misura per sviluppare in modo semplice ed efficace degli accurati sistemi di raccomandazione (il termine "Surprise" sta, appunto, per Simple Python RecommendatIon System Engine).
Il primo passo da compiere è stato, naturalmente, importare in Python le librerie da utilizzare:

```
from surprise import KNNBasic
from surprise import Dataset
```

[1] Add-ons per SciPy sviluppati in modo separato e indipendente dalla distribuzione principale.

```
from surprise import accuracy
from surprise.model_selection import train_test_split
```

Fortunatamente il dataset da utilizzare per le analisi ("Movielens 100K") era presente tra quelli contenuti in Suprise, e ciò mi ha permesso di caricarlo con l'ausilio di un semplice comando:

```
data = Dataset.load_builtin('ml-100k')
```

Successivamente, dal dataset precedentemente importato ho estratto un insieme di "training" e uno di "testing", facendo in modo che l'80% dei dati originari fossero utilizzati per addestrare l'algoritmo:

```
train, test = train_test_split(data, test_size=0.2)
```

Generati i set di "training" e "testing", sono passato all'implementazione dell'algoritmo vero e proprio. Come anticipato nella sezione 5.2.2, ho analizzato i risultati per tre valori di k (32, 50, 100) utilizzando tre misure di similarità: quella del coseno, l'indice di correlazione di Pearson e il Mean Squared Difference (MSD).
Per prima cosa, ho definito il tipo di misura della similarità da utilizzare (nell' esempio che segue, il MSD):

```
sim_options = {'name': 'msd', 'user_based': True}
```

Successivamente, ho utilizzato la misura precedentemente creata come argomento per la funzione di implementazione del k-NN (nell'esempio che segue, k=32):

```
kNN = KNNBasic(k=32, sim_options = sim_options)
kNN.fit(train)
```

Infine, ho testato la capacità predittiva dell'algoritmo e calcolato il RMSE come segue:

```
predictions = kNN.test(test)
accuracy.rmse(predictions)
```

A.2 SVD

Per l'algoritmo di Decomposizione ai valori singolari (SVD), similmente a quanto già illustrato per il k-NN, ho utilizzato Surprise, che permette di implementarne la celebre versione proposta da "Simon Funk" nel 2006 (per maggiori informazioni, si rimanda alla sezione 5.2.3).
Anche in questo caso, il primo step da seguire è necessariamente l'upload delle librerie da utilizzare:

```
from surprise import SVD
from surprise import Dataset
from surprise import accuracy
from surprise.model_selection import train_test_split
```

Analogamente a quanto visto in precedenza per il k-NN, ho caricato il dataset "Movielens 100K" e l'ho diviso in un insieme di "training" (80% dei dati totali) e in uno di "testing":

```
data = Dataset.load_builtin('ml-100k')
train, test = train_test_split(data, test_size=0.2)
```

Successivamente ho implementato l'algoritmo vero e proprio, che ho testato per tre livelli diversi del learning rate η (0.1, 0.01, 0.001), mantenendo invece costanti i valori corrispondenti al numero di fattori latenti (32) e al numero di epoche di addestramento (50).
Nell'esempio che segue mostro un'applicazione pratica di quanto detto in precedenza, per un valore del learning rate pari a 0.1:

```
SVD = SVD(n_factors = 32, n_epochs = 50, lr_all = 0.1,
```

```
biased = False)
SVD.fit(trainset)
```

Infine, ho effettuato le mie predizioni sul set di "testing" e calcolato il RMSE del modello come segue:

```
predictions = SVD.test(test)
accuracy.rmse(predictions)
```

A.3 Rete neurale "feed-forward"

Per implementare la rete neurale "feed-forward" ho utilizzato la nota libreria Keras [9], progettata appositamente per le applicazioni pratiche di deep learning. Nello specifico mi sono servito del backend[2] Theano, sviluppato con licenza open source dal LISA/-MILA Lab all' Università di Montreal.
In questo caso, per condurre la mia analisi sui dati, ho avuto bisogno di caricare altre librerie come Numpy, Scikit-learn e Pandas, molto utilizzate nel campo del Machine Learning e della data analysis:

```
import pandas
import numpy
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
```

Successivamente, ho utilizzato Pandas per caricare il dataset "Movielens 100K" in formato csv:

```
data = pandas.read_csv("path_to_file/u.data", sep =
 "\t", names = "userId,movieId,rating,timestamp")
```

[2]Librerie specializzate nella manipolazione di tensori, delle quali Keras si serve per operazioni come prodotti vettoriali e convoluzioni.

Inoltre, attraverso Numpy ho potuto convertire le variabili "userId" e "movieId" in categoriche:

```
data.userId = data.userId.astype("category").cat.code.values
data.movieId = data.movieId.astype("category").cat.code.values
```

Dal dataset così creato e pre-processato, ho potuto ricavare un insieme di "training" (80% dei dati totali) e di "testing" utilizzando Scikit-learn:

```
train, test = train_test_split(data, test_size = 0.2)
```

Una volta effettuate le operazioni precedenti, ho potuto costruire il modello di rete neurale "feed-forward".
Inizialmente, ho utilizzato una rete "shallow", con uno strato iniziale di "embedding", un unico strato nascosto "fully-connected" (contenente 200 neuroni) ed uno strato finale di output; ho inoltre fissato il "dropout" a 0.2, e il numero di fattori latenti per ambedue gli "embeddings" (userId e movieId) a 32.
Il seguente esempio mostra un'applicazione dell'algoritmo per un valore del learning rate η pari a 0.1:

```
#definizione delle variabili iniziali

n_users, n_movies(len(data.userId.unique()),
len(data.movieId.unique()))
n_factors_users, n_factors_movies = 32

#creazione degli embeddings

movie_input = keras.layers.Input(shape = [1], name = 'Movies')
movie_embedding = keras.layers.Embedding(n_movies + 1,
n_latent_factors_movie,name = 'Movies_Embedding')(movie_input)
movie_vec = keras.layers.Flatten(name = 'FlattenMovies')
(movie_embedding)
```

```
movie_vec = keras.layers.Dropout(0.2)(movie_vec)
user_input = keras.layers.Input(shape = [1], name = 'Users')
user_embedding = keras.layers.Embedding(n_users + 1,
n_latent_factors_user, name = 'Users_Embedding')(user_input)
user_vec = keras.layers.Flatten(name = 'FlattenUsers')
(user_embedding)
user_vec = keras.layers.Dropout(0.2)(user_vec)

#creazione della rete feed-forward

concat = keras.layers.merge([movie_vec, user_vec],
mode = 'concat',name = 'Merged_Layer')
concat_dropout = keras.layers.Dropout(0.2)(concat)
dense = keras.layers.Dense(200, name = 'FullyConnected_Layer',
activation='relu')(concat)
dropout_1 = keras.layers.Dropout(0.2, name = 'Dropout_Layer')
(dense)
result = keras.layers.Dense(1, activation = 'relu',
name = 'Activation_Layer')(dense)
opt = SGD(lr = 0.1)
model = keras.Model([user_input, movie_input], result)
model.compile(opt = optimizer,loss = 'mean_squared_error')
```

In seguito ho costruito in modo analogo una rete maggiormente "profonda" con tre strati nascosti "fully-connected", contenenti rispettivamente 200, 100 e 50 neuroni (ferme restando tutte le altre condizioni esposte in precedenza):

```
concat = keras.layers.merge([movie_vec, user_vec],
mode = 'concat', name = 'Merged_Layer')
concat_dropout = keras.layers.Dropout(0.2)(concat)
```

```
dense = keras.layers.Dense(200, name = 'FullyConnected_Layer')
(concat)
dropout_1 = keras.layers.Dropout(0.2, name = 'Dropout_Layer')
(dense)
dense_2 = keras.layers.Dense(100, name =
'FullyConnected_Layer2')(dense)
dropout_2 = keras.layers.Dropout(0.2, name = 'Dropout')
(dense_2)
dense_3 = keras.layers.Dense(50,name = 'FullyConnected_Layer3',
activation = 'relu')(dense_2)
result = keras.layers.Dense(1, activation = 'relu',
name = 'Activation_Layer')(dense_3)
opt = SGD(lr = 0.1)
model = keras.Model([user_input, movie_input], result)
model.compile(opt = optimizer,loss = 'mean_squared_error')
```

Una volta costruite le reti neurali descritte in precedenza, le ho addestrate sul dataset di "training" per un numero di epoche pari a 50:

```
model.fit([train.userId, train.movieId], train.rating,
epochs = 50)
```

Infine, ho calcolato il RMSE confrontando i rating predetti dal modello e quelli effettivi del dataset di "testing":

```
ratings_predict = numpy.round(model.predict[test.userId,
test.movieId], 0)
ratings_true = test.rating
mse = mean_squared_error(ratings_true, ratings_predict)
rmse = math.sqrt(mse)
```

A.4 LSTM

Per la Long-short term memory ho utilizzato Spotlight [10], una libreria basata su Pytorch che permette di implementare dei modelli sequenziali basati su reti neurali, tra le quali la LSTM.
Il modello, basato sull'articolo "Session-based recommendations with recurrent neural networks" [3], è stato costruito con poche e semplici righe di codice; come già visto per gli algoritmi precedenti, il primo passo è stato caricare tutte le funzioni necessarie ad implementare la LSTM:

```
from spotlight.sequence.implicit import ImplicitSequenceModel
from spotlight.sequence.representations import LSTMNet
from spotlight.cross_validation import
user_based_train_test_split
from spotlight.evaluation import sequence_mrr_score
from spotlight.datasets.movielens import get_movielens_dataset
```

Successivamente, ho caricato il dataset "Movielens 100K", contenuto all'interno della libreria, e l'ho suddiviso in un insieme di "training" (80% dei dati) e "testing", come mostrato di seguito:

```
data = get_movielens_dataset(variant = "100k")
train, test = user_based_train_test_split(data)
```

Per poter costruire il modello LSTM, è stata necessaria un'ulteriore operazione che trasformasse i set di "training" e "testing" in un formato "sequenziale", permettendo di ordinare le interazioni user/movie sulla base della data in cui si sono manifestate:

```
train = train.to_sequence()
test = test.to_sequence()
```

Una volta effettuate tutte le operazioni preliminari descritte in precedenza, ho potuto implementare l'algoritmo di LSTM, addestrando l'algoritmo per un numero di epoche

pari a 50, e utilizzando tre valori diversi del learning rate η (0.1, 0.01, 0.001).
L'esempio seguente mostra un'applicazione per un valore del learning rate pari a 0.1:

```
model = ImplicitSequenceModel(n_iter = 50, representation =
"lstm", loss = "bpr", batch_size = 64, learning_rate = 0.1)
model.fit(train)
```

Infine, ho analizzato il modello utilizzando come metrica il Mean Reciprocal Rank (MRR), come descritto nella sezione 5.3.2. Una volta calcolato il MRR per ciascuna sequenza di interazioni user/movie, ne ho calcolato la media aritmetica per valutare l'efficacia complessiva della LSTM:

```
mrr = sequence_mrr_score(model, test)
avgmrr = (mrr/len(mrr)).sum()
```

Bibliografia

[1] A. Krizhevsky, I. Sutskever e G. Hinton, *ImageNet Classification with Deep Convolutional Neural Networks*, Advances in Neural Information Processing Systems, 25, pp. 1097-1105, 2012.

[2] B. Furht, *Handbook of Social Network Technologies and Applications*, Springer-Verlag Berlin, Heidelberg, 2010.

[3] B. Hidasi et al., *Session-based recommendations with recurrent neural networks*, arXiv:1511.06939, 2015.

[4] B. Lantz, *Machine Learning with R*, Packt Publishing Ltd., Seconda Edizione, Luglio 2015.

[5] C. Basu, H. Hirsh e W. Cohen, *Recommendation as classification: Using social and content-based information in recommendation*, In: Proceedings of the Fifteenth National Conference on Artificial Intelligence, pp. 714–720, 1998.

[6] D. Bokde, S. Girase, D. Mukhopadhyay, *Matrix Factorization Model in Collaborative Filtering Algorithms: a Survey*, Procedia Computer Science, 49, pp. 136-146, 2015.

[7] D. E. Rumelhart, G. E. Hinton e R. J. Williams, *Learning representations by back-propagating errors*, Nature, 323, pp. 533–536, 9 Ottobre 1986.

[8] Descrizione ufficiale dell'algoritmo SVD nella variante di "Simon Funk", `http://sifter.org/simon/journal/20061211.html`.

[9] Documentazione ufficiale della libreria Keras, `https://keras.io/`.

[10] Documentazione ufficiale della libreria Spotlight, `https://maciejkula.github.io/spotlight/`.

[11] Documentazione ufficiale dello scikit Surprise, `http://surprise.readthedocs.io/en/stable/index.html`.

[12] D. Peppers, M. Rogers e Bob Dorf, *Is Your Company Ready for One-to-One Marketing?*, Harvard Business Review, January-February 1999 Issue.

[13] E. Gibney, *Google AI algorithm masters ancient game of Go*, Nature, 27 Gennaio 2016.

[14] F. M. Harper e J. A. Konstan, *The MovieLens Datasets: History and Context*, ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19, December 2015, 19 pages, DOI=`http://dx.doi.org/10.1145/2827872`

[15] F. Ricci et al., *Recommender Systems Handbook*, Springer-Science+Business Media, LLC, 2011.

[16] F. Rosenblatt, *The perceptron: A probabilistic model for information storage and organization in the brain*, Psychological Review, 65(6), 386-408, 1958.

[17] *Full length games for Go players to enjoy,* `https://deepmind.com/research/alphago/alphago-vs-alphago-self-play-games/`

[18] G. N. Yannakakis e J. Togelius, *Artificial Intelligence and Games*, Springer, 2018.

[19] I. Goodfellow, Y. Bengio e A. Courville, *Deep Learning*, MIT Press, 2016.

[20] I. Mackenzie, C. Meyer e S. Noble, *How retailers can keep up with consumers*, McKinsey & Company, 2013.

[21] J. A. Sánchez Mesa, C. Galán e C. Martínez, *The use of discriminant analysis and neural networks to forecast with a typical Mediterranean climate*, International journal of biometeorology, 49, 355-62, 2005.

[22] K. O'Shea e R. Nash, *An Introduction to Convolutional Neural Networks*, ArXiv e-prints, 2015.

[23] M. A. Nielsen, *Neural Networks and Deep Learning*, Determination Press, 2015.

[24] M. J. Pazzani, *A framework for collaborative, content-based and demographic filtering*, Artificial Intelligence Review, Volume 13, Issue 5-6, pp. 393–408, 1999.

[25] M. M. Evans, G. Foxall e A. Jamal, *Consumer Behaviour*, Wiley, 2009.

[26] N. Craswell, *Mean Reciprocal Rank*, In: L. Liu, M.T. Özsu (eds), Encyclopedia of Database Systems, Springer, Boston, MA, 2009.

[27] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, *Dropout: a simple way to prevent neural networks from overfitting*, The Journal of Machine Learning Research , Volume 15 Issue 1, pp. 1929-1958, January 2014.

[28] P. Cotter e B. Smyth, *PTV: Intelligent personalized TV guides*, In: Twelfth Conference on Innovative Applications of Artificial Intelligence, pp. 957–964, 2000.

[29] P. Melville, R. J. Mooney e R. Nagarajan, *Content-boosted collaborative filtering for improved recommendations*, In: Proceedings of the Eighteenth National Conference on Artificial Intelligence, pp. 187-192, Edmonton, 2002.

[30] P. Melville e V. Sindhwani, *Recommender Systems*, In: Sammut C., Webb G.I. (eds) Encyclopedia of Machine Learning and Data Mining, Springer, Boston, 2017.

[31] *Recommender Systems 101*, `https://d4datascience.wordpress.com/category/predictive-analytics/`.

[32] R. Jozefowicz, W. Zaremba e I. Sutskever, *An empirical exploration of recurrent network architectures*, Proceedings of the 32nd International Conference on International Conference on Machine Learning, Volume 37, pp. 2342-2350, Lille, 2015.

[33] S. Cheng, *The awful frustration of a teenage Go champion playing Google's AlphaGo*, Quartz Media (qz.com), 27 Maggio 2017.

[34] S. Haykin, *Neural networks and learning machines*, Pearson Education, Upper Saddle River, NJ, Terza Edizione, 2009.

[35] S. Hölldobler, S. Möhle e A. Tigunova, *Lessons Learned from AlphaGo*, 2017.

[36] Sito ufficiale del Premio Netflix, `https://www.netflixprize.com/`.

[37] T. Chai e R. R. Draxler, *Root mean square error (RMSE) or mean absolute error (MAE)? – Arguments against avoiding RMSE in the literature*, Geosci. Model Dev., 7, pp. 1247-1250, `https://doi.org/10.5194/gmd-7-1247-2014`, 2014.

[38] T. Chouard, *The Go Files: AI computer wraps up 4-1 victory against human champion*, Nature, 15 Marzo 2016.

[39] V. Bhatnagar, *Collaborative Filtering Using Data Mining and Analysis*, IGI Global, 2016

[40] X. He, L. Liao, H. Zhang, L. Nie, X. Hu e T. Chua, *Neural Collaborative Filtering*, arXiv:1708.05031v2, August 2017.

[41] Y. Koren, R. Bell e C. Volinsky, *Matrix Factorization Techniques for Recommender Systems*, Computer, Volume 42, Issue 8, August 2009.

Dipartimento di Impresa e Management

Corso di Laurea Magistrale in Marketing

Cattedra di Machine Learning & object driven Marketing

Riassunto dell'elaborato: "Reti neurali e collaborative filtering: un approccio avanzato al marketing one-to-one"

A cura di: Pasquale De Rosa, matr. 686081

Anno Accademico 2017/18

Capitolo 1

Sistemi di raccomandazione e marketing one-to-one

Il presente capitolo ha discusso le generalità dei sistemi di raccomandazione, uno degli strumenti più avanzati e innovativi applicati nell'ambito del marketing one-to-one.
Un sistema di raccomandazione mira a semplificare il complesso processo di "decision making" da parte di un consumatore fornendogli, nel preciso tempo e spazio in cui si verifica l'insorgenza del bisogno, suggerimenti su specifici prodotti/servizi da acquistare.
Nello specifico, sono tre le tipologie di sistemi di raccomandazione illustrate nell'elaborato in oggetto, ossia le tecniche di "collaborative filtering", gli algoritmi di "content-based filtering" e gli approcci "ibridi" tra le due "famiglie" descritte precedentemente.
Per "collaborative filtering" si intende un approccio alla raccomandazione basato sulla convergenza di interessi tra utenti con gusti simili; in ragione di ciò, se un particolare prodotto ha incontrato in passato il gradimento di uno specifico cliente, sarà possibile "consigliare" l'acquisto dello stesso ad altri utenti, che presentino delle rilevanti affinità con il primo profilo.
I sistemi di raccomandazione appartenenti a questa tipologia presentano, come facilmente intuibile, un significativo numero di vantaggi strutturali, come ad esempio la

possibilità di suscitare interesse, in particolari utenti, verso prodotti "nuovi" o in quel momento sconosciuti, oppure la notevole flessibilità ed efficienza computazionale; di contro, però, il loro utilizzo può determinare l'insorgenza di taluni svantaggi, come il c.d. problema della "partenza a freddo", ossia la scarsa efficacia di questi metodi laddove l'oggetto della raccomandazione siano prodotti di nuova immissione sul mercato, o l'estrema "sparsità" delle informazioni relative ai giudizi/ratings attribuiti da ciascun utente ai prodotti commerciati dall'impresa.

Gli algoritmi di collaborative filtering sono solitamente suddivisi in due macrocategorie fondamentali: i c.d. metodi "neighborhood-based", basati sul concetto di "nearest neighbors", ossia i k utenti più "simili" al consumatore cui è rivolta la raccomandazione, e sulla base dei quali l'intero sistema è strutturato, e le c.d. tecniche "model-based", che poggiano invece su un complesso sistema di strumenti statistici per "predire" i giudizi attribuiti a un prodotto da parte di uno specifico utente, fermo restando l'impianto "collaborativo" della raccomandazione in oggetto.

Per "content-based filtering" s'intende, invece, un approccio alla raccomandazione maggiormente "personalizzato", basato unicamente sulle preferenze esplicitate in passato da uno specifico utente.

Anche in questo caso, come per il collaborative filtering, l'utilizzo di tale "famiglia" di strumenti presenta dei notevoli vantaggi, come ad esempio l'eliminazione del problema della "partenza a freddo", o la possibilità di effettuare raccomandazioni indipendenti per i singoli utenti, senza necessità di analizzare il comportamento di altri clienti simili ai primi; tuttavia, anche in questo caso, è possibile citare taluni difetti strutturali, come ad esempio la scarsa capacità predittiva di questi modelli in presenza di nuovi utenti, o l'impossibilità di suggerire prodotti o servizi ulteriori rispetto a quelli solitamente utilizzati da ciascun cliente in portafoglio.

Negli ultimi tempi si è assistito a un significativo incremento dell'utilizzo di metodologie "content-based" basate su recensioni e altre forme di dati non strutturati, come i documenti testuali: un esempio è dato dal c.d. "Vector Space Model" (VSM) che tende a rappresentare ciascun documento in un dato "corpus" come un vettore n-dimensionale

i cui elementi sono i termini contenuti nella recensione. Ampiamente utilizzati nelle applicazioni pratiche risultano, inoltre, le tecniche di classificazione mutuate dall'apprendimento supervisionato, come ad esempio il c.d. "classificatore bayesiano naif", o "Naïve Bayes", un algoritmo derivato dal noto teorema statistico del calcolo delle probabilità condizionate, noto anche, per l'appunto, come "teorema di Bayes".
In aggiunta ai sistemi di raccomandazione di tipo "collaborativo" e "content-based" si è affermato, negli ultimi tempi, anche l'utilizzo di talune tipologie "ibride" tra le due famiglie di algoritmi descritti in precedenza: a tal proposito, si sono susseguiti, nella letteratura esistente in materia, diversi contributi rilevanti, tra i quali si può ricordare il "Content-boosted collaborative filtering", proposto da P. Melville et al., o il "Collaboration via content", elaborato nell'ambito di uno studio condotto da M. J. Pazzani.
Ciascuna delle tecniche di raccomandazione ivi descritte costituisce, come affermato in apertura di capitolo, un notevole strumento di "marketing one-to-one", ossia quell'approccio al marketing teso alla creazione di relazioni estremamente "personalizzate" con ciascun cliente presente in portafoglio, allo scopo di incrementare significativamente il valore economico che ciascuno di questi legami ha, durante l'intero "ciclo di vita" delle interazioni stabilite con il consumatore.

Capitolo 2

Le reti neurali artificiali: tipologie e generalità

Le sezioni successive hanno descritto le caratteristiche generali delle c.d. "reti neurali artificiali", introducendone al contempo le tipologie principali, che godono di maggiore utiizzo nelle applicazioni pratiche in materia.

Una rete neurale artificiale è un algoritmo di apprendimento automatico la cui struttura replica, in estrema sostanza, l'aspetto di una rete neurale biologica. Proprio come il neurone "umano", infatti, il neurone artificiale è composto da tre elmenti fondamentali, ossia i dendriti (che ricevono gli impulsi in entrata dagli altri elementi della rete), il corpo cellulare (che rappresenta il fulcro dell'attività cerebrale, elaborando gli impulsi elettrici in entrata e in uscita) e le terminazioni assoniche (che "propagano" l'output del corpo cellulare nei successivi strati della rete).

Una prima forma di neurone artificiale fu proposta, nel 1958, dallo psicologo americano Frank Rosenblatt, e prese il nome di "percettrone"; il percettrone originario non è altro che un sofisticato algoritmo di classificazione, il quale, partendo da un numero n di input $x_1, x_2, ..., x_n$, restituisce un unico output binario (0 o 1 a seconda dell'appartenenza o meno delle osservazioni alla categoria in esame).

Nello specifico, la classificazione nel modello proposto da Rosenblatt è eseguita confrontando la somma degli n input, "ponderata" attraverso un egual numero di pesi, $\omega_1, \omega_2, ..., \omega_n$, con un "valore soglia" t, in modo tale che:

$$\text{Output} = \begin{cases} 0, \text{se} \sum_n x_n \cdot \omega_n \leq t \\ 1, \text{se} \sum_n x_n \cdot \omega_n > t \end{cases} \tag{2.1}$$

Tale espressione rappresenta anche le condizioni di "attivazione" del neurone artificiale, e per tale ragione è spesso definita con la notazione di "funzione di attivazione", dove, nel caso specifico in oggetto, la funzione utilizzata è anche nota con il nome di "funzione di Heaviside".

Nella letteratura vigente in materia non è infrequente l'utilizzo di un ulteriore parametro, noto come "bias" del percettrone (b), in sostituzione del "valore soglia" t; nello specifico, essendo $b = -t$, la (2.1) può essere riscritta come:

$$\text{Output} = \begin{cases} 0, \text{se} \sum_n x_n \cdot \omega_n + b \leq 0 \\ 1, \text{se} \sum_n x_n \cdot \omega_n + b > 0 \end{cases} \tag{2.2}$$

La funzione di Heaviside non rappresenta l'unica tipologia di attivazione utilizzabile per la costruzione di un modello di rete neurale; un ulteriore approccio, che gode di notevole applicazione empirica, prevede l'impiego di una funzione "sigmoidea", la cui espressione analitica è la seguente:

$$\sigma = \frac{1}{1 + e^{-\sum_n x_n \cdot \omega_n + b}} \tag{2.3}$$

La funzione di attivazione sigmoidea possiede un vantaggio notevole, se confrontata con quella di Heaviside: essendo continua tra 0 e 1, infatti, non rende il proprio output particolarmente suscettibile a forti variazioni nei pesi e nei bias, come invece si verifica per l'altro modello esposto in precedenza.

Il percettrone e il neurone "sigmoideo" rappresentano dei modelli di rete neurale estremamente semplici, in quanto costituiti da un unico "strato" di input, comprendente i valori $x_1, x_2, ..., x_n$, e un unico valore di output, rappresentante l'esito della predizione o classificazione operata dall'algoritmo; applicazioni più recenti hanno "ampliato" la struttura di tali modelli, attraverso l'introduzione di livelli intermedi tra lo strato di input e quello di output, noti anche come "strati nascosti".
Uno strato nascosto contiene neuroni che applicano trasformazioni sui dati di input, allo scopo di permettere la propagazione del maggior quantitativo di informazioni possibili verso lo strato finale; in particolare, le reti neurali possono essere dotate di un solo strato di questa tipologia (e in tal caso sono definite anche reti "shallow") o di un numero maggiore di strati intermedi (e a tal scopo viene ad esse attribuito l'appellativo di reti "profonde"). In generale, queste particolari architetture di rete neurale sono note anche con il nome di "percettroni multistrato".
In aggiunta ai modelli sovracitati, sono state introdotte nel tempo diverse tipologie di reti neurali "complesse", tra le quali è doveroso citare:

- Le c.d. "reti convoluzionali", le quali, in virtù della loro particolare struttura, risultano essere estremamente adatte per l'elaborazione di input costituiti da dati "grafici", per loro natura tridimensionali, i quali sono propagati verso lo strato finale della rete (attraverso complesse procedure come le "convoluzioni" e il "max pooling") senza che si verifichi una perdita delle informazioni originarie;

- Le c.d. "reti neurali ricorrenti", note anche come reti "feed-back" in quanto capaci, a differenza di ciascuno dei modelli illustrati in precedenza, di "modellare" adeguatamente dati di natura sequenziale, mantenendone inalterate le dinamiche temporali alla base.

Capitolo 3

Metodologie per l'addestramento di reti neurali

Il capitolo in esame ha analizzato le principali metodologie algoritmiche utilizzate ai fini dell'addestramento di reti neurali artificiali, illustrando le tecniche maggiormente diffuse, in dottrina, per la selezione dei parametri del modello, ossia i pesi $\omega_1, \omega_2, ..., \omega_n$ e i bias b.

Più specificamente, è stato introdotto il concetto di "funzione di costo" come unità di misura della perdita connessa ad uno specifico modello (nel caso in oggetto, una rete neurale), la cui rappresentazione generica è la seguente:

$$C = f(\omega, b) \tag{3.1}$$

Nella pratica, sono molteplici le funzioni costo che possono essere utilizzate per "quantificare" l'efficacia predittiva di una rete neurale (all'interno di questa tesi, ne sono state analizzate tre nello specifico, ossia il "mean-squared error", il "mean-absolute error" e la "cross-entropy"), ma ciascuna di esse è accomunata da due caratteristiche fondamentali:

- Assumono necessariamente valori ≥ 0;

- Tendono a 0 quanto più il valore della predizione associata al modello si avvicina al dato realmente osservato nel "testing set".

Nel dettaglio, la seconda condizione precedentemente enunciata suggerisce che un generico modello di rete neurale possieda una maggiore efficacia predittiva laddove la funzione di costo ad esso associata valga, approssimativamente, 0; ciò significa che, ai fini di una selezione ottimale dei parametri ω e b che descrivono il modello, è necessario individuarne i valori che determinino una minimizzazione della funzione di costo $C(\omega, b)$.
Ciò può essere ottenuto attraverso una metodologia nota, in letteratura, come "discesa del gradiente"; questo algoritmo, come suggerisce il nome, si basa sul concetto di "gradiente" ∇C di una funzione di costo, ossia il vettore delle sue derivate parziali rispetto ai due parametri del modello, $(\frac{\partial C}{\partial \omega}, \frac{\partial C}{\partial b})$.
Nello specifico, l'algoritmo di discesa del gradiente prevede che la variazione complessiva della funzione di costo possa essere computata come:

$$\Delta C \approx \nabla C \cdot (\eta \cdot \nabla C) \tag{3.2}$$

Dove η è un parametro, definito "learning rate", che misura l'"intensità" con cui ci si muove all'interno del gradiente della funzione di costo, o, per usare altri termini, la "velocità" con cui lo si "discende" (utilizzando valori negativi di η) alla ricerca del minimo globale di C.
Nelle applicazioni pratiche, avviene non di rado che, nel caso in cui la funzione di costo C possa essere "scritta" come media o somma di più funzioni differenziabili, ciascuna rappresentante la perdita C_x associata a uno specifico input del modello, sia utilizzata una "variante" dell'agoritmo standard di discesa del gradiente, nota come "discesa stocastica del gradiente" (SGD), che permette di velocizzare notevolmente le computazioni necessarie a trovare il minimo di C eseguendo le operazioni descritte in precedenza su gruppi di "mini-batch", ossia piccoli sottoinsiemi degli n input originari della rete.

Come si può agevolmente desumere dalle precedenti osservazioni, ai fini della scelta dei parametri ottimali di un modello di rete neurale riveste un ruolo centrale il corretto calcolo del gradiente della funzione di costo ∇C; nella pratica, la determinazione del gradiente è il risultato di un procedimento complesso, noto come "algoritmo di back-propagation", il quale, basandosi su una nota formula matematica nota come "regola della catena", permette di stimare ($\frac{\partial C}{\partial \omega}$ e $\frac{\partial C}{\partial b}$), per ciascuno strato l della rete, come segue:

$$\frac{\partial C}{\partial \omega_{jk}^{l}} = a_{k}^{l-1} \cdot \delta_{j}^{l} \tag{3.3}$$

$$\frac{\partial C}{\partial b_{j}^{l}} = \delta_{j}^{l} \tag{3.4}$$

Dove, rispettivamente, a_{k}^{l-1} è la k-esima attivazione presente nello strato precedente, $l-1$, e δ_{j}^{l} esprime l'errore relativo al j-esimo neurone all'interno dello strato l, ed è anch'esso calcolabile applicando la regola della catena.

L'algoritmo di backpropagation, pur risultando ad oggi la metodologia maggiormente utilizzata, in combinazione con l'algoritmo di discesa del gradiente, per l'addestramento delle reti neurali artificiali, ha mostrato una sostanziale inefficacia in presenza di reti di natura "ricorrente". Nello specifico, nel caso dei modelli "feed-back" si è osservato come il gradiente computato attraverso l'applicazione della regola della catena fosse sostanzialmente "instabile", tendendo a produrre stime inaccurate dei parametri ω e b, soprattutto negli strati iniziali della rete.

Per tale ragione, diverse "architetture" di rete neurale ricorrente sono state proposte in letteratura al fine di ovviare al problema dell'instabilità del gradiente; di queste, la più utilizzata nelle applicazioni empiriche risulta essere la c.d. "Long-short term memory" (LSTM), un complesso modello "feed-back" in cui le informazioni sono "immagazzinate" all'interno di piccole strutture, definite "cell states", permettendo la gestione efficace di un grande quantitativo di dati sequenziali anche in presenza dell'utilizzo dell'algoritmo di backpropagation.

Capitolo 4

Collaborative filtering: un confronto empirico tra algoritmi "standard" e reti neurali

Le sezioni di cui si compone il presente capitolo hanno descritto dettagliatamente i risultati del mio lavoro di ricerca individuale, condotto allo scopo di confrontare gli esiti dell'applicazione di modelli "neurali" per la costruzione di un sistema di raccomandazione di tipo "collaborative filtering", con i "findings" ottenuti utilizzando approcci maggiormente "tradizionali", quali il k-Nearest Neighbors (k-NN) e la Decomposizione ai valori singolari (SVD).

A tal scopo è stato utilizzato un dataset specializzato, noto come "Movielens" (nella sua variante "100K"), il quale raccoglie 100000 ratings, in scala che va da 1-5, attribuiti da 943 utenti a ben 1682 film, in un arco temporale complessivo di sette mesi (da Settembre 1997 ad Aprile 1998).

Relativamente al k-NN, è stato analizzato l'andamento di un indicatore misurante la

perdita generata dal modello, noto anche come "root-mean squared error" (RMSE), al variare di due parametri fondamentali: la misura di "similarità" utilizzata per identificare i k "neighbors" più vicini a ciascuna osservazione nel dataset (nello specifico, all'interno del lavoro in oggetto ne sono state proposte tre, ossia il "mean-squared difference", la "similarità del coseno" e l'"indice di correlazione di Pearson"), e il parametro k, per il quale sono stati utilizzati tre valori (32, 50, 100), allo scopo di individuare quello associato al miglior risultato, in termini di capacità predittiva, generato dal modello.

Gli esiti della ricerca in questone hanno evidenziato che, utilizzando un valore $k = 32$ e il mean-squared difference come misura di similarità, fosse possibile ottenere il miglior risultato predittivo per l'algoritmo k-NN, in corrispondenza di un valore del RMSE pari a 0.9813.

Un analogo studio è stato effettuato relativamente alla tecnica di Decomposizione ai valori singolari, o SVD, nella "variante" proposta nel 2006 da un partecipante al celebre "premio Netflix", noto con lo pseudonimo di "Simon Funk"; nello specifico, utilizzando l'algoritmo di SGD per l'addestramento del modello, per valori del learning rate η pari a 0.001, 0.01 e 0.1, è stato possibile ottenere il risultato migliore in corrispondenza di $\eta = 0.001$, con un valore finale del RMSE pari a 0.9809, inferiore a quello generato attraverso l'applicazione dell'algoritmo k-NN.

In una fase successiva, sono stati proposti due modelli di rete neurale "feed-forward", le cui capacità predittive sono state confrontate con quelle dei due algoritmi "tradizionali" illustrati in precedenza. Nello specifico, ambedue le reti in esame si componevano di uno strato di "embedding", uno di "flatten" e uno di "dropout", differendo in virtù del diverso numero di strati nascosti "fully-connected" presenti al loro interno; ciò era dovuto, essenzialmente, al fatto che per costruire la prima rete, di tipo "shallow", ne fosse stato utilizzato soltanto uno, laddove, di contro, per la rete maggiormente "profonda" fosse possibile rinvenirne tre, prima di pervenire allo strato finale di output.

Anche in questo caso, come per il modello di SVD descritto in precedenza, è stata utilizzata la discesa stocastica del gradiente per l'addestramento delle reti, adoperando

valori del learning rate η pari a 0.001, 0.01 e 0.1; gli esiti della ricerca in tal modo condotta, hanno evidenziato, per ambedue i modelli di rete neurale proposti, i risultati migliori in corrispondenza di $\eta = 0.01$, con valori del RMSE pari, rispettivamente, a 0.9624 per la rete "shallow", e 0.9593 per quella maggiormente "profonda".
Infine, è stato condotto un ulteriore studio proponendo un modello di rete "sequenziale" di tipo LSTM, ai fini della costruzione di un sistema di raccomandazione che tenesse conto delle strutture temporali insite nei dati di "training"; gli esiti di questa ricerca, tuttavia, hanno evidenziato la sostanziale inefficacia di tale approccio, attraverso l'analisi di un indice sintetico noto come "mean reciprocal rank" (MRR), che ha presentato valori estremamente bassi in corrispondenza di ciascun parametro del learning rate η utilizzato per la fase di addestramento del modello.
In virtù di quanto precedentemente esposto, è legittimo concludere affermando che le reti neurali "feed-forward" possano essere considerate, ad oggi, tecniche decisamente all'avanguardia nella costruzione di sistemi di collaborative filtering di comprovata efficacia, generando predizioni maggiormente accurate rispetto agli algoritmi tradizionalmente destinati a tal scopo, ossia k-NN e SVD; inoltre, le reti maggiormente "profonde", ossia dotate di un numero più elevato di strati intermedi "fully-connected", presentano i migliori risultati in tal senso, ponendo le basi, nel lungo periodo, per un complessivo rinnovamento delle soluzioni legate al marketing personalizzato "2.0", basato su tecniche e strumenti mutuati dal "deep learning".